1-1-2014

Software Power Analysis And Optimization For Power-Aware Multicore Systems

Shinan Wang
Wayne State University,

SOFTWARE POWER ANALYSIS AND OPTIMIZATION FOR POWER-AWARE MULTICORE SYSTEMS

by

SHINAN WANG

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2014

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor Date

_____

_____

_____

DEDICATION

To my beloved family.

ACKNOWLEDGMENTS

First of all, I would like to express my deepest appreciation to my advisor, Prof. Weisong Shi, who provides invaluable support and visionary guide to my five years Ph.D program in Wayne State University. His persistence in the path of seeking the truth inspired me when I was frustrated. He is not only a knowledgeable mentor for academia, but also a kind and generous elder who impacts my career. With his continuous research enthusiasm for more than 10 years, the MIST lab been widely known for original ideas and unconventional research fruits under his supervision. I am also very grateful to Prof. Fisher, Prof. Brockmeyer, and Prof. Jiang, who serve as the committee members and give valuable suggestions on the prospectus and finally finishing the dissertation. They showed me what an excellent researcher, advisor, and educator could be. I am also thankful to Dr. Arnetz and Dr. Wiholm for their kindly support on SPA project and help for viewing problems other than from computer science. In addition, I am also thankful to all my colleagues in the MIST Lab, LAST group, especially Guoxing Zhan, Tung Nguyen, Hui Chen, who broadened my horizon by viewing problems from different aspects. Guoxing set an excellent example for the MIST group and I hope his spirit will inspire every generation of the group. I am also thankful to Dr. Kewei Shi, who showed me the first lesson of how to conduct research projects when I first joined the program. Last but not least, I deeply appreciate the support from my mother (Yufang Yao) and father (Xiaofeng Wang) and my ex-wife (Xuan Li).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this chapter, we will first introduce the motivation of this work. Objectives are described in details, followed by the contributions and the outline of the rest of the thesis.

1.1 Motivation

Sustainable Computing is an application of the political concept of sustainability to the IT world [96]. In sustainable computing, designers of computer systems, ranging from small mobile devices to massive data centers, emphasize obtaining a sustainable level of environmental and societal costs as a first-order design principle. These cost may result from the computer systems manufacturing, operation, and disposal.

Among all the factors in sustainable computing that contribute to system operational costs, power dissipation and energy consumption are fundamental in modern computer systems [112]. Their effects can be found across computational domains, including data center design [94], enterprise level server design [93], battery life management on a smart phone [23, 102, 113], and circuit layout on a microprocessor. As an indispensable component of a computer system, software has a profound impact on power dissipation [52]. The number of workloads running on computer systems grows rapidly. For example, there are more than 500,000 applications available for download on iOS in 2011 [140]. Nonetheless, the effects of software on efficient system design is unclear and the challenges are presented as follows:

First of all, different from conventional performance metric, power dissipation is difficult to measure because hardware instrumentation is usually required; thus, the prerequisite for power efficient design is an accurate and verbose power estimation [45, 85, 114, 147]. A better understanding of the power dissipation of a system will enable more power-saving opportunities [66]. However, most existing approaches do not expose sufficient information to end-users

and software developers [5, 12, 28, 74, 123, 143]. Hence, information scarcity of dynamic power dissipation impedes the progress of efficient software design. The challenge is that there is a gap between the power dissipation of hardware and the applications running on it. On the other hand of the spectrum, high performance is a *sufficient but not necessary condition* to high power dissipation. Even though power models estimate power dissipation of a system, the optimization points is unclear to the application developers. Poorly designed code sections have high power dissipation but do not necessarily produce high performance [116]. As a result, energy is wasted. Based on this observation, we envision software power optimization has two-fold: static and dynamic. *Statically*, with more detailed information on the causes of power dissipation of workloads [131], software developers will be able to leverage algorithms and implementations to achieve better energy efficiency. Obviously, software developers are the best candidates who can identify inefficiency in a piece of software. Tuning applications for power and energy saving or energy-performance tradeoff is a major concern for power-aware computing. *Dynamically*, fine-grained power management posts challenges on software power behavior analysis in details. The subtle relationship between software and hardware resource usage eliminates the possibility of mentioning each one of them without considering the other. In this dissertation, we undertake a novel approach that profiles the workload power dissipation at a fine-grained level. By leveraging this technique, we optimizes to improve the energy efficiency of a software based on a new metric, and involves software in the process of power management schemes of the operating systems.

1.2 Objectives

Given the challenges listed above, our long term objective is to integrate software into the loop of efficient system design, which exposes software power dissipation information to underling operating systems. Our short term goal is to investigate the enabling technologies for efficient software/workload design and fine-grained power management.

1. *Analyze component power dissipation of a computer system.* The first step in the study is to break down the power dissipation of a computer system, which gives a detailed overview of static power vs. dynamic power for each computer component. The rest of the objectives are based on the findings of our first step.

2. *Investigate power dissipation in association with workload execution.* In order to get the detailed profiling information, not only we will model the power dissipation of the major components in a system, but also develop a mechanism to distinguish the power dissipation of each execution phase in the software.

3. *Automate the process of workload power analysis.* In order to analyze power dissipation of workload in practical situations, an accuracy power model is inadequate. The profiling process needs to generate minimized instrumentation overhead and scalable for large software.

4. *Model relationship between energy consumption of a workload and the system configurations using the power profiling techniques described above.* While power analysis provides runtime workload power dissipation information, different system configurations change power dissipation and energy consumption in a subtle way.

5. *Design and implement a workload aware mechanism to achieve energy efficiency.* The lack of information of workload and computer architectures results in poor efficiency. System power management schemes need to consider the behavior of running software/-workloads. Furthermore, power management schemes need to be adaptive if the behavior varies. We propose to model parallel program in terms of C (Concurrency), P (Power dissipation), and T (Execution time) to achieve better overall system energy efficiency. The expected output of this part is a power management scheme that adjusts system components mode, such as DVFS and concurrency level, based on workload and system characteristics.

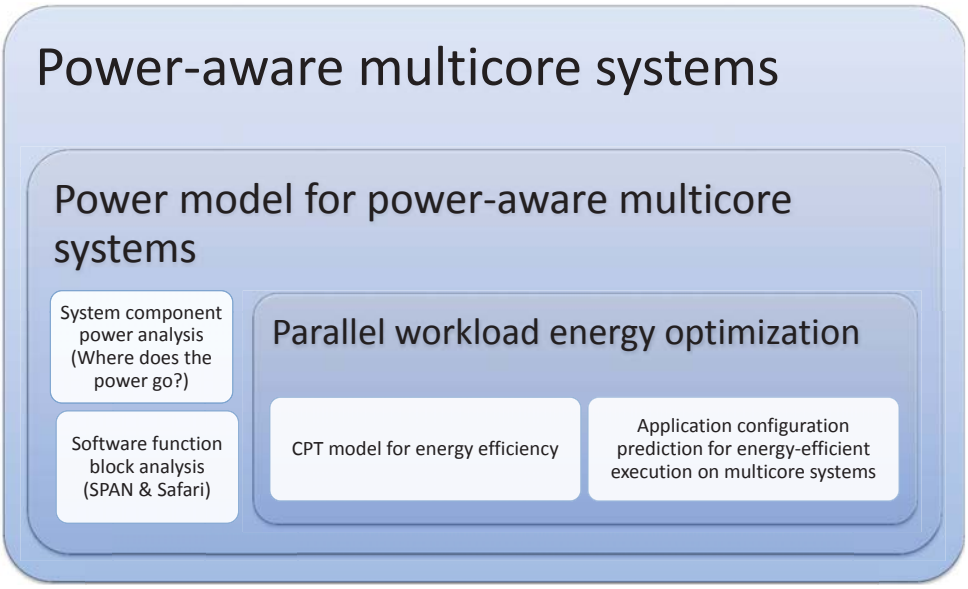


Figure 1.1: Overview of our approach.

1.3 Our Approach

Over all, our approach can be divided into four parts, as described in Figure 1.1. We will describe each section of it in the following sub sections.

1.3.1 Component power dissipation analysis

In this part of the thesis, in order to first understand modern computer system design in terms of power usage, we analyze two computer systems that have been built over periods. The power dissipation of the computer system or a single computer device includes two parts, the static power and the dynamic power. The static power of a computer device is the power dissipated on this device when it is in idle state; the dynamic power of a computer component is the extra power dissipated on this device when it is active. More specifically, we define the idle state of disk as when it is spinning but no data access operations. The sum of the static power and the dynamic power is the total power dissipated by this component. Similarly, the static power of the computer system is the basic power needed to maintain the running of the system when the system is idle; the dynamic power is the extra power needed when executing the tasks. More clearly, we define the idle state of the computer system as when the percentage

of CPU utilization is about zero and all the other components are in idle state.

Static power accounts for a very large ratio of the total power dissipation of the computer system. When the system is idle, most of the energy used in this state is not considered as used for computing. Thus, one important task of power management is try to decrease the static power of the components and the computer system. For example, by using clock gating most sub-units of CPU can work in the low power mode when it is not used. In this way, the static power of the CPU is decreased. In addition, new memory refresh strategy [126] makes the memory could be refreshed with lower power compared with memory access operation. All of these methods try to decrease the waste of energy generated by static power. To decrease the dynamic power, people try to design low power circuits and use new energy saving materials, such as phase-change memory (PCM) [82]. PCM uses a special kind of non-volatile storage material which do not need to refresh to maintain the data in the memory. Software method aims to improve the energy usage efficiency by improving the executing efficiency and making different power saving strategies for the system.

Although, the aforementioned work have been done in power management area, the computer systems are still consuming an ever increasing amount of energy and the power dissipation of computer system do not decrease too much. *Where does the power go in a computer system* is a question that grabs more and more people's interests but have not yet been answered clearly. It is important to realize the power profile of the computer system. Thus, people get to know which area deserves more research work. Moreover, *what is the trend of power management of the computer system?* Finally, *whether there are some implications, which can serve as guidance for future research?* The goal of this part of the work is attempt to answer these questions. First, we give a clearly view of the power profile in the computer systems. Second, through the experiment we find some observations, which tell us the trend of power management in the last several years. Finally, we find some implications that are helpful for future research. For example, CPU utilization can not accurately reflect power dissipation of

the CPU. We define CPU utilization as the time spent in executing the task set, as opposite to the time spent to execute idle task, such as I/O wait.

1.3.2 Power analysis and modeling for power-aware multicore systems

In this subsection, our primary goal is to model and analyze power dissipation of a computer system in association with the resource usage information. Many previous efforts have focused on computer system power measurements and profiling [10, 15][28, 46, 71, 81, 84]. Actually, power dissipation of a single computer system can be broken down into several pieces with each piece representing a component, such as CPU or memory.

Furthermore, power dissipation of each component consists of two parts: static power and dynamic power. The former could be described as the basic power supplied to maintain this component in its operational state. The latter is the additional power dissipation for running a specific task. For years, it has been well-acknowledged that dynamic power is roughly determined by utilization rates, especially for CPUs. However, the experimental results show that, for the CPU dynamic power, the estimation error rate of using this method can be as high as 33.33% [27]. On the other hand, understanding the power dissipation behavior of a specific software/application is the key to write efficient software and design energy-efficient computer systems. Therefore, we need a more accurate model to capture the power dissipation of computer systems.

Usually, there are four ways to estimate power dissipation: *cycle-level system simulators*, *instruction-level modeling*, *software-function-level macro-modeling*, and *PMCs-based modeling*. Cycle-level system simulators are time costly while providing more detailed information [17, 147]. Instruction level modeling achieves simplicity and accuracy on embedded systems, but it is not realistic if we apply it to superscalar processors with a large number of instructions. For example, IA-32 ISA contains 331 different instructions, with 109561 (331^2) instruction combinations if considering the inter-instruction effects. Software-function-level macro-modeling techniques associate power dissipation with application function sub-routines

and establish power models on top of application characteristics, such as algorithm complexity [129]. However, such information sometimes is inherently unavailable for end users. Moreover, the static feature of this method prevents its utility when we consider advanced run-time power management. Analytical power modeling based on performance monitoring counters (PMCs) enables run-time software power estimation [28, 72, 81, 107]. Nevertheless, for a given processor, the power model based on PMCs is limited by the types of available event counters and the maximum number of counters that can be read simultaneously. For instance, most Intel processors only support sampling two counters per core concurrently. Regarding the power estimation utilizing PMCs, however, we also need to notice that accuracy highly depends on two sets of PMCs: those PMCs appearing in power models and those PMCs available on targeting platforms. Insufficient information representing the power characteristics of the microarchitecture will yield low accuracy.

Software contributes to a considerable portion of the total power of a computer system [27, 8, 116]. Hence, it is very important to find out how much power has been dissipated by a specific software component in order to design sustainable computer systems. Power dissipation, arguably speaking, is a fundamental aspect of software nowadays. On one hand, the total energy consumption of completing a task is power accumulation over time. Thus, power dissipation is a direct contributor to producing an energy profile. On the other hand, in some particular circumstances, controlling power dissipation provides more flexibility for systems. For example, temperature can be altered by restricting power dissipation. Besides, some infrastructures add ”power envelop” as one of the constraints. For instance, it is crucial for data centers serving millions of people to maintain the whole power budget under a certain limit for power supply protection (huge current draw may damage transistors). As a result, it is worth to investigating the run time power dissipation of an application and the associated source code for sustainable computing point of view. We focus our discussion on identifying run-time factors that determine the power dissipation of processors for computation intensive workloads on

power-aware multicore computer systems. Concretely, we model power dissipation in a two-level manner to reserve simplicity and accuracy. More importantly, we map power dissipation to software blocks at runtime by building SPAN libraries and interfaces. Specifically, the work presented in this paper includes the following contributions.

1.3.3 Source code power dissipation profiling

Power dissipation of a piece of software is a basic property that needs software developers to detect, tune, and optimize. By designing the proposed function-level power profiling technique, we are able to detect software "hotspots" [25], which contains power intensive code and optimize it as a software developer if necessary. Usually, the higher the performance metrics a code section has, the more power-hungry it is. However, this is not always the case. Nevertheless, some hotspots have negative effects towards the operating machine, such as causing high temperature. The results, however, do not reflect the causes of high power dissipation. For example, while high CPU utilization will definitely lead to high power dissipation, the cause not necessarily is poorly written code. On the opposite, this method could be known as HUGI (hurry up and go idle) or race idle. Thus, it is reasonable to reveal the real causes of high power dissipation from the software point of view and optimize the code section showing poor power efficiency. The benefits of the proposed research include the follows: first of all, we are able to reveal power efficiency of a piece of software application at a fine-grained level. In addition, the tool we design will be able to make suggestions on power deduction according to the power efficiency of different section of code [103]. We will deploy a two level analysis approach, which becomes full-fledged function level power analysis tools that integrate hardware activity indicators. The input contains various operating system statistics, such as the contents under procfs in UNIX-like operating systems. The output expected is a list of software functions that are estimated to have high power dissipation. We will utilize PMCs as major resources for such information. Example inputs are LLC misses, hard page faults et al. At the first stage, we will focus on general purpose computers and then optimize

for resources-constricted systems. In order to get the detailed profiling information, we will develop a mechanism to distinguish the power dissipation of each function in the software. We design SPAN, which specifies a set of APIs to be inserted into the source code. Alternatively, to enable automatic source code instrumentation, we will utilize compiler techniques to insert profiling code before and after each function in source code. The expected outcome includes an open source function level power profiling tool, Safari. SPAN and Safari estimate function level power dissipation based on power models.

1.3.4 Software/workload energy-efficient configuration selection

Given the estimated power behavior of software, it is urgent to develop an on-the-fly approach of software-behavior-aware power management scheme. There is a gap between power management schemes of a operating system and the software application (workload) running on it. The existing power management approaches mainly speculate the behavior of a workload, which usually generate either inaccurate or coarse guidance toward the operating system. If the behavior of a workload are profiled in advance, we are able to achieve more timely and fine-grained power management schemes. We will utilize the tools that extending from SPAN [137] to generate profiling information about benchmark software with detailed information about its resources usage and power dissipation information. The data are mainly from PMCs. At runtime, the profiling data from the target software are compared with a pre-collected data which contains matrices defining possible power saving opportunities for DVFS, thread mapping, et al. Specifically, we model the energy efficiency as three tuples, C (Concurrency), P (Power dissipation), and T (Execution time) for a parallel workload, namely, CPT model. Base on the model and runtime profiling information, the system will select the optimal system configuration for the target software.

1.4 Summary of contributions

In the summary, the main contributions of this dissertation are as follows:

1. We describe our power measurement method in detail. In addition, we provide a fine-grain power profile of the computer system. Then, we study the trend of power management of the computer system in the last several years. Finally, we derive out eight implications from our observations, which are important to the energy efficient system design in the future.

2. We propose a two-level power model for power-aware multicore computer systems. The novelty of the proposed model is two-fold. First, we minimize the number of performance counters and training benchmarks utilized in the model to achieve simplicity and applicability. Second, we incorporate frequency in the power model to meet the requirements of modern DVFS techniques. The experimental results based on *SPEC*2008*C jvm* benchmark suite show the average error rate of 5.40% across one core to six core validation.

3. We design and implement SPAN to relate power dissipation to the different portions of an application using the proposed power model. By using SPAN, developers can easily identify the sections of code consuming the most power in the program. Alternatively, to enable automatic source code instrumentation, we utilize compiler techniques to insert profiling code before and after each function in source code. The expected outcome includes an open source function level power profiling tool, Safari. SPAN and Safari estimate function level power dissipation based on power models. The experiment results show that Safari is able to produce function level profiling with limited overhead (on average 16%) and controlled estimation error of 6.85% on average. Additionally, we apply software power profiling on a modern MIPS architecture based multicore processor. Along with CPU, we have developed power model for memory and coprocessors. Practically, we have proven the generality of our function level power profiling on other platforms.

4. We propose CPT model, a energy-efficient model to capture the relationship between concurrency (C), power(P), execution time(T) and workload energy efficiency. Three case studies are used to demonstrate the usage of CPT model.

5. Based on the CPT model and power profiling technique we propose, we design mechanism to capture the optimal energy-efficiency for parallel workload. Execution information using two threads is used to predict the energy consumption of different configurations on a specific architecture. We use a DVFS mechanism to adjust CPU frequency according to the workload information during the run-time given the predicted concurrency level and thread mapping setting. The experimental results based on a Xeon E5620 server with NPB and PARSEC benchmark suites show that the model is able to predict the energy efficient configuration accurately for 100% tested benchmarks. An additional 10% EDP saving is obtained by using run-time DVFS on average for the entire system.

1.5 Outline

The rest of this document is organized as follows: Chapter 2 reviews the related work; Chapter 3 analyzes the current trend of power dissipation of computer systems by systemically measure and profile two general purpose computers; Chapter 4 proposes a power analysis and modeling for power-aware multicore systems and SPAN - a software function level multicore processor power analyzer. Chapter 5 describes the profiling tool Safari, which automates the profiling process; In Chapter 6, we propose a workload energy efficiency model, CPT model, for parallel workload by using our profiling tool; By using CPT model, we locate the optimized system configuration in terms of energy consumption of a workload on a multicore processor system in Chapter 7; Chapter 8 gives the conclusion of this thesis discusses the future work. this research and the author's publication record.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter describes the related work in each aspect: computer system power measurement & profiling and energy-efficient system design. In the first part, we briefly summarize the research on the computer system power measurement, especially hardware instrumentation so that the readers can have a general understanding of current techniques. The second section introduces the latest research on energy-efficient system design. With such a brief background introduction, the readers will have a comprehensive understanding of this dissertation.

2.1 Power Measurements and Profiling

In this sub section, we describe the basic research concepts in power measurement and hardware instrumentation.

As energy consumption becomes one of the foremost considerations in designing new computer systems, power-aware system design raises a key issue in the community of computer systems. Power measurement and profiling, which are the basis of power-aware systems, not only can be used to evaluate power optimization techniques and to make power-performance trade-off, but also can be used to generate critical power information for operating systems and power-aware software. Based on hardware and software techniques used, power measurement and profiling could be classified into three categories: *hardware-based method*, *software-based method*, and *hybrid method*. Table 2.1 summarizes the classification of these previous efforts.

2.1.1 Hardware-based Method

The hardware-based methods mainly use two strategies: using meters to build a power measurement and profiling platform or integrating power sensors into hardware architectures.

Direct Power Measurement and Profiling

The first strategy uses meters to measure the currents or voltages of wires that supply power for hardware, and then compute power dissipation with these result. This strategy is usually used to evaluate the accuracy of software methods. In this thesis, we also use this classic power measurement method to evaluate the accuracy of our models. One of the earliest studies of power measurement and profiling is done by Viredaz *et al.* on handheld computing devices [135]. Joseph *et al.* use a similar method to measure the power dissipation on a high performance processor [71, 72]. They use a group of microbenchmarks with particular cache, bit activity, and branch prediction behaviors to evaluate performance and power trade-off. In [75], the authors adopt the direct power measurement method; then, they measure the power on a Cray XT4 supercomputer under several HPC workloads. Their results show that computation-intensive benchmarks generate the highest power dissipation. Nevertheless, memory-intensive benchmarks yield the lowest power usage. Physical measurement is fast and objective, but this method lacks a semantic connection between measurement results and evaluated programs [61].

Integrate Power Sensors into the System

The second strategy is usually used by high-performance servers [48, 29]. For example, Intel uses service processor-based power-monitoring sensors to provide power information for systems through the API called Intelligent Platform Management Interface (IPMI) [64, 48]. IBM BladeCenter and System x_TM servers supply PowerExecutive solutions that enable customers to monitor actual power draw and temperature loading information [29]. Though online power-aware applications can use this method, it is difficult to yield low-level power information because hardware circuits are too complicated to distinguish the originality of power dissipation. In addition, power monitoring circuits also dissipate a large amount of power as well.

2.1.2 Software-based Method

Even though hardware-based methods are more accurate than software-based methods, hardware cost and scalability requirements restrict their application range. In addition, during an architecture design cycle we cannot use hardware-based method to balance power and performance. Software-based methods use power models to estimate power dissipation. Power models are created at different levels: circuits level, instruction level, component level, node level, and so forth. Based on different usage stages, we summarize software-based methods into two types: architecture-level power models, which are used to estimate power dissipation during the architecture design stage, and system-level power models, which supply live power information to operating system and power-aware applications.

Architecture-level Power Model

Software-based methods spring up in the area of architecture-level power estimation. Most of the earliest work [85, 91, 17, 147, 100] in this category are based on the classic energy equation [68]. Liu *et al.* estimate the power on VLSI CMOS chips [85]. Register transfer level power model is analyzed by Marculescu *et al.* in [91]. Brooks *et al.* proposed Wattch, a framework for analyzing and optimizing microprocessor power dissipation at the architecture-level [17]. The power model of Wattch relies on per-cycle resource usage counts. In [147], Ye *et al.* present a comprehensive framework called SimplePower, which is based on the transition sensitive energy models. It not only can be used to evaluate the effect of high-level algorithmic, architectural, and compilation trade-off on energy, but also provides the energy consumption in the memory system and the on-chip buses using the analytical energy models. SoftWatt, which models the CPU, memory hierarchy, and the low-power disk subsystem, is described in [52]. This tool is able to identify the power hot-spots in system components as well as the power-hungry operating system services.

The power constraints in interconnection network design were noticed by[100]. Also, this

paper proposes the power model of routers and links and analyzes the performance of direct interconnection network topologies under a fixed power constraint. Wang *et al.* present a power-performance interconnection network simulator called Orion, which is capable of providing detailed power and performance characteristics to enable rapid power-performance trade-off at the architecture-level [136]. Eisley *et al.* estimate and analyze the power of CMPs by synergistically considering both the processor cores and the communication fabric in a multi-core chip [37]. Chen *et al.* propose SimWattch to integrate the system-level and the user-level simulators [26].

Besides those efforts that model the power dissipation of processors, several publications [17, 149, 146, 53, 97] propose methods to estimate the power of other devices, such as hard disks, memories, and network devices. Zedlewski *et al.* present Dempsy, a disk simulation environment that includes the accurate modeling of the disk power dissipation [149]. Dempsey attempts to estimate the power of a specific disk stage, which includes seeking, rotation, reading, writing, and idle-periods, with a fine-grained model. Molaro *et al.* also analyze the possibility to create a disk driver power model based on disk status stages [97]. In [53], the authors build the power model for hard disk based on the observation that a slight change on the rotation speed of a disk has a quadratic effect on its power dissipation. In [146], Ye *et al.* introduce a framework to estimate the power dissipation on the switch fabrics on network routers and propose different modeling methodologies for the node switches, internal buffers, and interconnect wires inside switch fabric architectures.

System-level Power Model

Specialized circuit techniques are important strategies for low-power designs, but these techniques alone are not sufficient. Higher-level strategies for reducing power dissipation and improving energy efficiency are increasingly crucial [17]. The live power information of systems is highly needed for designing high-level energy efficiency strategies. For example,

Ecosystem [150] and [89], which propose the concept managing system energy as a type of resource, require the support from real-time power information on different levels. Furthermore, in [134], the authors argue that the traditional operating system design should be revisited for energy efficient usage. As part of the energy-centric operating system, energy profiles are also needed by new power-aware scheduling algorithms [3, 77]. Ahmad *et al.* propose a new power-aware scheduling algorithm based on game theory [3]. In [77], Khan *et al.* present a method, which is also based on game theory, to minimize the energy consumption on computational grids. System-level power models are built on the statistics of systems, which reflects activities of the hardware devices.

One of the earliest research in this category is [131]. Tiwari *et al.* propose an instruction-level power model for embedded processors and memories. Russell *et al.* present an energy model using a constant parameter for power dissipation of a 32 bit embedded processor [114]. T. Li and L. Kurian John [84] exploit a high correlation between the instruction per cycle (IPC) and the power dissipation, and they predict the run-time power dissipation on the OS routines based on regression model between power and IPC. G. Contreras and M. Martonosi [28] also discover the power-IPC correlation and use five PMCs to estimate the power of workloads running on different CPU frequencies. Their model exhibits low percentage of error, but they do not verify the model on multicore architectures. Bircher *et al.* [15] explore the run-time events that most likely represent power dissipation. In their experiments, IPC-related metrics are shown to be the most power-informative. Among those metrics, the upos fetched per cycle yields the most accurate results. Other candidates are upos completed per cycle and upos retried per cycle. Wu *et al.* [143] also use a number of PMCs to deploy a power model on the Pentium 4 functional units. They measure the CPU power via a clamp-on ammeter. However, their model is not validated under different frequencies and multicore architecture.

Dhiman *et al.* [34] propose an on-line power prediction system on virtualized environments. Instead of using linear regression models on PMCs, they utilize a Gaussian Mixture

vector quantization based training and classifying. The estimation error is within 10% in most cases. Bertran *et al.* [12] demonstrate an alternative approach using PMCs on the CPU power estimation. Rather than directly deriving power models using PMCs, they propose a method to treat each component of the CPU separately, such as FE, INT, and FP. Combining all the training parameters, they develop a fine-grained power model. However, the training process is too time-consuming to be extensively used in practical situation. In addition, the power model highly depends on the microarchitecture of the CPU. Bellosa [10] demonstrates the correlation between recorded performance events and the power dissipation from the synthetic workloads. He shows the most effective factors of system power dissipation are: fuops/sec, uops/sec, L2 accesses/sec, and memory accesses/sec. Because he only considers the synthetic workloads, the results could not be sufficiently applied to real applications.

In [106], Powell *et al.* propose a methodology to reduce the number of performance counters while maintaining certain accuracy of the model. They estimate the hardware activity events of several microarchitectural structures; then, the authors associate the activity events to the power dissipation of such structures. Singh *et al.* [123] describe an approach based on a number of microbenchmarks which stress the particular components of a given processor architecture. Our work differs from all these works in the way that we combine CPU frequency scaling and multicore features in the power model, which fits the trend of microprocessor design recently.

2.1.3 Hybrid Method

Hybrid methods are also globally researched [42, 67, 46] because both hardware-based and software-based methods have their own limitations. Flinn *et al.* develop a platform that samples both the power dissipation and the system activities on a profiling computer; then, they generate an energy profile from the data through an off-line analysis [42]. Isci *et al.* build a platform using sampled multimeter data for overall power measurements and produce per-unit power breakdowns based on the hardware performance counter readings [67]. Their

power model uses 22 performance monitoring counters and reaches as low as 5% error rate on the SPEC 2000 benchmarks. However, the large number of performance counters may not be available for sampling simultaneously on some processors. For example, most Intel platforms only support concurrent sampling of two counters. In this case, to retrieve the information from 22 counters, the program has to be run at least 11 times. Ge *et al.* develop a power measurement and profiling platform to retrieve the power information from the main components, such as the CPU, disk, memory, motherboard, and so forth [46]. Also, they propose a method to map the measured power into the application code and analyze the energy efficiency in a multi-core system. Isci *et al.* develop an experimental framework to compare the control-flow based with the performance-monitoring-based power-phase detection techniques [65]. Their results show that both the control-flow and the performance statistics provide useful hints of the power phase behaviors.

Chang *et al.* rely on statistical sampling to help programmers evaluate the energy impact of their design decisions [25]. In [61], they describe an evaluation infrastructure, which combines the advantages of simulations and physical measurements for the OS/compiler power and energy optimizations. In addition, this infrastructure can provide the objective evaluation and semantic connection between the measured power/energy and the source code. Lorch *et al.* design two programs: PowerMeasure, which is used to measure how much power each component consumes in predefined state, and StateProfiler, which is used to profile how often each component stays in a specific power state [87, 88].

2.2 Energy-Efficient Design

Given a brief background description of power measurement and profiling, we will discuss energy efficient design in this section. Energy-efficient design though, is not a new topic. Various techniques are presented to control consumed energy. Roughly speaking, each technique fits in one of the following categories regardless of platform.

Hardware-based	Software-based	Hybrid
Itsy00 [135]	Wattch [17]	PowerScope [42]
Jpseph01 [71]	SimplePower [147]	Isci03 [67]
Kamil08 [75]	SoftWatt [52]	PowerPack [46]
PowerExecutive [29]	Orion [136]	Chang03 [25]
IMPI [64]	SimWattch [26]	Lorch97 [87]
	Dempsey [149]	
	Bellosa00 [10]	
	vEC [74]	
	powell:2009 [106]	
	Bertran10 [12]	

Table 2.1: Classification of Power Profiling Efforts.

2.2.1 Energy Conservation on Conventional Computer System

In this section, we review the earliest work have been done to design energy-efficient computing systems. Taking performance as the first priority, energy-wise design was less considered at that time. The research were mainly focusing on CPU, disk, and display on portable devices.

One of the famous paper that opened continuous work to reduce CPU energy usage, was written in 1994 [139]. A new concept is introduced as millions-of-instructions-per-joule (MIPJ). The core idea is dynamic controlling the clock speed. However, the energy consumption for a particular job does not decrease since the MIPJ required remains the same. The real benefit comes from reduced voltage while the clock slows down. One consequence affects the performance to apply such technique, is the extended execution time. Given the bottom line is to save energy, the possibility of scheduling tasks at different CPU cycle time is examined. This work has greatly inspired the development of Dynamic Voltage Scaling (DVS) and numerous paper has published targeting toward this field.

Encouraged by the benefits, researchers start building theoretical model [145] based on the previous work. The objective is finding the most energy-efficient way to schedule the tasks with the guarantee that all deadlines are met. In this work, based on the assumption

that power is associated with the CPU speed as a convex function, how the scheduling of jobs affect the overall energy consumption is analyzed primarily in a off-line mode. This model has been widely used because its simplicity and soon became a guideline for energy-efficient CPU scheduling.

On-line heuristic scheduling of aperiodic tasks while retaining the feasibility of periodic task sets is presented in [56]. Non-preemptive power aware scheduling is proposed in [55]. Another method tries to slow down the CPU whenever there is a single task eligible for execution was developed in [122]. A more aggressive approach is presented in [6], where both offline and online algorithms are considered to meet deadlines while reduce the cycle speed as much as possible. A systematic comparison of different scheduling algorithms on the delay vs. performance trade-off is demonstrated in [50]. In [104], deadline information are adopted in the real-time operating system along with the DVS technique. To sum up, most of these proposed scheduling algorithms attempt to leverage the energy-efficiency and the timing constraints of a real time system.

For the alternative approach, rather than direct developing scheduling algorithms for operating system, is optimizing the program at the very beginning. In [60], a a compiler algorithm is designed targeting toward effectively optimizing programs for energy usage using dynamic voltage scaling (DVS). Similar work can be found in [59]. Basically, the objective is to identify the CPU voltage scaling chances without ruining the performance significantly while in the compilation time. Though this direction of research shows some different angles on efficient usage of CPU, the fundamental idea is similar. The essence is to adaptively tuning the voltage with the reasonable compromised performance.

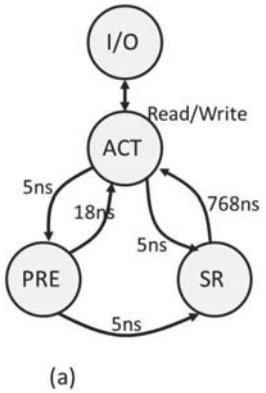
For the software strategies for the purpose of energy saving, Lorch and Smith (1998) suggest heuristics [88] to

- (i) avoid running processes that are still blocked and waiting on an event;
- (ii) delaying processes that execute without producing output or signaling useful activity;

- (iii) delaying the frequency of periodic processes that are not likely to produce useful services.

Compared with the research interest put into CPU usage, other components in the computing system are supported only by few studies. As the benefit provided by different working modes on processors, the industry realized the importance of integrating this mechanism in different devices. The simulation results show that the overall energy consumption of the system is not as optimistic as what has been done on CPU solely, especially when interactions to the memory are needed [92]. As a result, there were voices that DVS can be applied on memory management as well. This observation is confirmed by the fact that the benefit from DVS is diluted on the embedded systems with low-power processor and standard memory [105]. [39] illustrates by simulation that neither memory power management nor the DVS techniques on processor can save energy dramatically. But by combining two technique together, totally 89% energy saving, compared with standard base case, can be made. Totally three combination are examined in the experiments. First, the memory power is constant over the entire period with the standard memory, so the lowest energy is achieved by minimizing the CPU energy. Another one is naive power awareness memory that can power down at the end of the period and the completion of the task. The results show that it is no longer best choice to extend the execution as long as possible to minimize the CPU energy consumption. On the opposite side, the lowest CPU frequency produces the highest energy in this case. While the dynamic power-aware is the advanced technology that can use lower power while the task executing, which is also known as aggressive policy. The overall energy saving is maximized using this policy, particularly at the lower frequency during which the memory can power down. All the above experiments are based on the MPEG decoding program and demonstrate the power management on memory helps to realize effects of DVS.

A memory power management techniques are proposed in [141] with page migration to group active pages in close ranks in a memory system as much as possible so that the rest of the



	Power State	Idle Power(W/GB)
Active Standby	ACT	1.34
Precharge Fast Powerdown	PRE	0.7
Self-refresh	SR	0.23

(b)

State transition	Energy (nJ/GB)
PRE →ACT	19.3
SR →ACT	583.7

(c)

Figure 2.1: Memory System Architecture. (figure courtesy of [141])

ranks are able to enter lower power state. This mechanism relies on the power saving between different power states and execution slacks to minimize power usage. However, transition overhead between different power states needs to be considered as Figure 2.1 shows. RamZzz is integrated into PTLSim v3.0 simulator [148] for validation.

The basic optimization policies and algorithms mentioned before have focused on predicting the opportunities to switch the whole memory or part of it into low power mode, either at the run-time, or during the compilation process. The former is the hardware-assisted approach that can decide the idle time at the cycle level while execution by dynamically analyze the workload on memory. Whereas, the later is based on statistical analysis or called heuristic approach which attempts to identify the possible chances to slow the memory frequency.

As the major efforts have been made on the CPU and Memory power management, another group of researchers focuses on the distributed and networked systems. As a result, the wireless networking protocols and interfaces are becoming increasingly intensive studied, especially when the mobile devices spread all over. L. Feeney and M. Nilsson experiments the wireless characteristic in the Ad-Hoc network in 2001 [41]. The results are briefly explained in Figure 2.2.1. For example, the item (a) means a point to point transmission while (b) means a broadcast sending operation. The linear coefficients is determined by measurements. Usually, energy consumption is modeled by the summation of a fixed cost, which is associated with the

	$\mu W * sec/byte$ $\mu W * sec$
point-to-point send (a)	1.9 × *size* + 454
broadcast send (b)	1.9 × *size* + 266
point-to-point recv (c)	0.50 × *size* + 356
broadcast recv (d)	0.50 × *size* + 56
	non-destination n ∈ S, D
promiscuous recv (e)	0.39 × *size* + 140
discard (f)	-0.61 × *size* + 70
	non-destination n ∈ S, n ∉ D
promiscuous recv (g)	0.54 × *size* + 66
discard (h)	-0.58 × *size* + 24
	non-destination n∉ S, n ∈ D
promiscuous "recv" (i)	0.0 × *size* + 63
discard (j)	0 × *size* + 56
idle (ad hoc) (k)	843mW
idle (BSS)	66mW

Table 2.2: Model of power measurement. (table courtesy of [41]

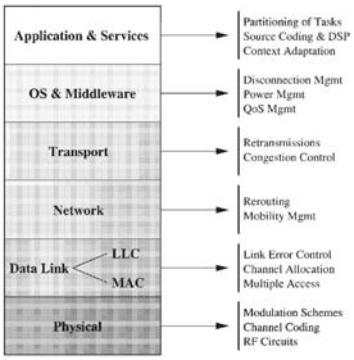


Figure 2.2: Protocol stack of a generic wireless network, and corresponding areas of energy efficient research. (figure courtesy of [70])

working type, and incremental cost, which relates to the size of data receiving or sending. This model is simple enough to estimate the energy consumption of wireless communication.

The networking itself is a layered architecture that contains Physical Layer, MAC Layer, Network Layer, Transport Layer, OS and the application [70]. Each level has different guideline for power management, such as routing protocol in the network layer, channel allocation in the MAC layer, and so on. Figure 2.2 illustrates the networking architecture and the basic energy-efficient schemes that used within that layer.

The above mentioned three areas are considered to be worth noticing, however, a number of topics raise from a different angles to achieve the energy-efficient design as well which can

not be throughout covered. This section attempt to convey the idea that energy saving design can be obtained from any part of the system.

For example, most efforts in energy-efficient communication design among system components have focused on the lower layers of the stack including the physical and link layers. The basic idea behind these approaches is to encode the binary data sent through the communication channel to minimize its average switching activity, which is proportional to dynamic power consumption. Ramprasad et al. (1998) studied data encoding for the minimum switching activity problem and obtained upper and lower bounds on transition activity reduction for any encoding algorithm [109]. The main idea proposed by these approaches is encoding the binary data stream sent through the communication channel when possible to reduce the number of switching activity, which affects the power consumption.

2.2.2 Studies on System Level of Energy Saving

The research community soon realized the significant impact of overall system level strategy. Luca Benini and Giovanni De Micheli (2000) classifies the system components consuming major portion energy into three category: *computation units; communication units; and storage units* [11]. They argue that the energy-efficient design in a part of the system (e.g., the computing element) can affect others (e.g., memory and/or I/O). Amin Vahdat, et al. (2000), provide an overview of what they envisioned in the energy-efficient design for post-PC applications in the new century [134]. Since the processors becoming powerful, the memory growing huge, along with the increasing bandwidth, battery capacity is improving at a modest pace. Adding the energy saving into one of the functionality of operating system, undoubtedly increases the complexity of the system design. However, the authors suggest to explore the energy-efficiency in the following aspects: resource management, communications, and remote computation. Though the techniques the proposed might be similar others in each part of the system, the overall system energy optimization is the goal rather than for a particular part.

To be more specific, for example, memory instructions are among the more power-hungry operations on the embedded systems and are well studied. Therefore, it can be a proper example of resource management. Both a static hardware policy and dynamic hardware policy are developed based on the appearance of Direct Rambus DRAM (RDRAM) [108], which implements totally four power state as shown in Figure 2.3. The novel of their research is the "sequential first-touch" which allocates the pages as the order they are accessed. In this way, most relevant pages are placed together as many as possible so that the rest of the memory can be in low power state without ruining the performance much. In regard of communications, there are also numerous opportunities for power optimization including *i*)adjusting the transmission power based on the distance of receiver and sender,*ii*)redesign the routing protocol for the energy-balance or the minimum energy consumption purpose. In addition, the networking communication, as they argue, can be a cross-layer design based on the application demand. For example, the time sensitive communication may minimize latency, while others may minimize the power. The last point mentioned is how to leveraging remote computation. The trade-off can be made between sending the data to remote server for computation and locally execute the program. As a more application-oriented aspect, decisions must be made case by case.

Another example advocate system level power management can be found in the [150]. The fundamental contribution is defining the unified $CurrentcyModel$ accounting over various hardware devices and enable reasonable energy allocation among competing applications. A unit of currentcy represent the right to consume a certain amount of energy during a fixed time period. The biggest issue in the modeling is how to represent the energy requirement for each device. First, the authors deduct the currentcy accounting for the CPU, disk, and network card. For example, the cost of an disk access is computed as

$$\frac{active-state-power-cost(W)}{disk-access-bandwidth(KB/s)} * buffersize(KB)$$

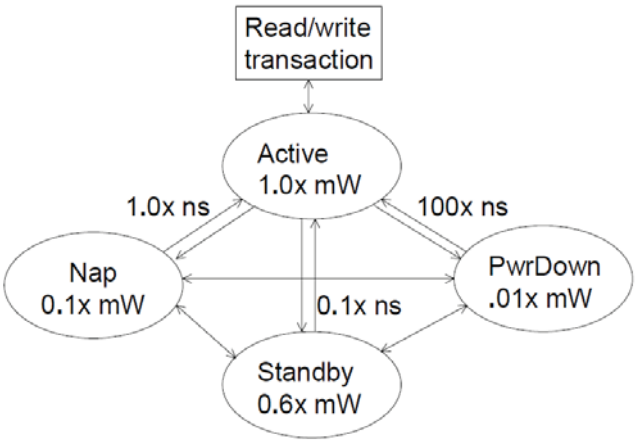


Figure 2.3: RDRAM Power States. (figure courtesy of [134])

In addition, the cost of spinning up and down the disk is shared by all tasks, so does the spin up. Regarding the network interface, sending and receiving energy are calculated as follows:

$$E_{send} = (sent\_bits * transmit\_power)/bit\_rate$$

$$E_{recv} = (received\_bits * transmit\_power)/bit\_rate$$

The detail modeling can be a tedious and tricky task. The idea is to estimate the energy one operation consumes on a particular device. The primary goal of ECOsystem is achieving a target battery life, which determines how much total currentcy can be allocated in each energy epoch. As a result, the total energy consumption during a time period is under control by suspending the energy-greedy task. The allocating policy is mainly determined by the priority of the task or the user defined configuration. Table 2.2.2 demonstrate the energy sharing for two tasks, ijpeg and Netscape. Under a fixed total 5W energy, by allocating portion of it to different task, the performance can vary, though the target battery life can be achieved.

As the aforementioned examples shows, the new energy-efficient design trend are mov-

Energy Share	ijpeg			Netscape		
	Power Alloc(W)	Age Power Used(W)	CPU Util(%)	Power Alloc(W)	Ave Power Used(W)	Page Load Latency(sec)
70%:30%	3.5	3.507	22.55%	1.5	1.49	29.205
60%:40%	3.0	3.008	19.43%	2.0	2.006	17.441
50%:50%	2.5	2.500	16.08%	2.5	2.457	9.928
40%:60%	2.0	2.008	12.91%	3.0	2.961	6.322
30%:70%	1.5	1.503	9.67%	3.5	3.443	3.934
20%:80%	1.0	1.005	6.46%	4.0	3.663	3.032

Table 2.3: Proportional Sharing: ijpeg vs. netscape, 5W Total Energy. (table courtesy of [150])

ing from optimizing a particular component in a system to the overall perspective and from general purpose computer system to embedded systems. Being provided by various energy-efficient features from the hardware devices, the operating system are expected to flexibly manipulate these characteristics. The challenges in this area include how to compare the energy consumption of two different devices(e.g, CPU and memory), how to control the device behavior from the operating system, and how to leverage the competing requests for the energy from different applications. Another trend can be easily observed is the application-oriented energy conservation scheme, or even the user-centric energy configurations generated from each user's behavior. For example, a remote computation scheme can be applied to the CPU intensive applications. Whereas, the data compression or/and the page allocation schemes are helpful when the application is data-centric.

CHAPTER 3

WHERE DOES THE POWER GO?

We start the research topics in this dissertation by providing a detailed power dissipation measurement and analysis of two computer systems. Specifically, in this chapter, we use two computers of different period, one is an old computer about five years ago (PC_{05}) and another one is a new computer (PC_{10}), to measure the power dissipation of the main components and make a comparison between them. We use direct an indirect method to measure the power when the system is idle or running different kinds of software benchmarks. Through the analysis of the experiment result, we answer the questions we proposed.

3.1 Introduction

In the last several years, power dissipation of computer systems and the subsequent problems, such as energy security, environment and climate change, are gradually invoking more and more concerns. Many works in circuits design, hardware architecture design and software implementation have been done so as to decrease the energy consumed by computer systems. For example, several energy saving techniques, such as clock gating technique [79, 99], dynamic voltage scaling (DVS) [20, 119] and dynamic frequency scaling (DFS, also known as clock throttling) [119], are used by the CPU. In addition, software methods, which try to improve the energy efficiency of computer systems, are also researched globally. The Green Grid group proposed the definition of power usage effectiveness (PUE) [51], aiming to improve the power efficiency and decrease energy wasted in data centers. In [126], they find the mismatch between the workload and the power dissipation in data centers. Nowadays performance is not the only consideration anymore when design the computer system. How to decrease the power dissipation of computer systems becomes the foremost issue that people both in academic and industrial areas try to solve.

The power dissipation of the computer system or a single computer device includes two parts, the static power and the dynamic power. The static power of a computer device is the power dissipated on this device when it is in idle state; the dynamic power of a computer component is the extra power dissipated on this device when it is active. More specifically, we define the idle state of disk as when it is spinning but no data access operations. The sum of the static power and the dynamic power is the total power dissipated by this component. Similarly, the static power of the computer system is the basic power needed to maintain the running of the system when the system is idle; the dynamic power is the extra power needed when executing the tasks. More clearly, we define the idle state of the computer system as when the percentage of CPU utilization is about zero and all the other components are in idle state.

Static power accounts for a very large ratio of the total power dissipation of the computer system. When the system is idle, most of the energy used in this state is not considered as used for computing. Thus, one important task of power management is try to decrease the static power of the components and the computer system. For example, by using clock gating most sub-units of CPU can work in the low power mode when it is not used. In this way, the static power of the CPU is decreased. In addition, new memory refresh strategy [126] makes the memory could be refreshed with lower power compared with memory access operation. All of these methods try to decrease the waste of energy generated by static power. To decrease the dynamic power, people try to design low power circuits and use new energy saving materials, such as phase-change memory (PCM) [82]. PCM uses a special kind of non-volatile storage material which do not need to refresh to maintain the data in the memory. Software method aims to improve the energy usage efficiency by improving the executing efficiency and making different power saving strategies for the system.

Although, the aforementioned work have been done in power management area, the computer systems are still consuming an ever increasing amount of energy and the power dissipa-

tion of computer system do not decrease too much. *Where does the power go in a computer system* is a question that grabs more and more people's interests but have not yet been answered clearly. It is important to realize the power profile of the computer system. Thus, people get to know which area deserves more research work. Moreover, *what is the trend of power management of the computer system?* Finally, *whether there are some implications, which can serve as guidance for future research?* The goal of this chapter is try to answer these questions. First, we give a clearly view of the power profile in the computer systems. Second, through the experiment we find some observations, which tell us the trend of power management in the last several years. Finally, we find some implications that are helpful for future research. For example, CPU utilization can not accurately reflect power dissipation of the CPU. We define CPU utilization as the time spent in executing the task set, as opposite to the time spent to execute idle task, such as I/O wait.

The remainder of this chapter is organized as follows: Section 3.2 describes the background of our work and related works of this chapter. Then, we will describe the method we used to measure the power of these main components in Section 3.3. In Section 3.4, we relates the configuration of our experiment platform, how do we make the experiment and the evaluation of the experiment result. Section 3.5 will talk about the implications we get from the analysis of the experiment result. Then, we will make the conclusion in Section 3.6. Finally, Section 3.6 talks about the future work of power management.

3.2 Background & Related Work

While several previous work [8, 110, 18] have addressed the problem of energy unproportionately in the computer systems, but none of these works tell the fine-grain power dissipation of the computer systems. And that no people have made a comparison between old computer systems and the new computer systems. In [8], they find that the power efficiency in a data center is low, and that the energy used for computing usually accompanied with a large amount of energy that are not really used for computing, for example cooling down the computer systems.

[110] gives an global view of the sources of waste of energy in the computer systems, also it gives a bunch of recipes for energy efficiency in computer systems. Finally, [18] tells us the strong requirement of power management in computer system with a bunch of statistical data. It describes the power management strategies used by software method. More importantly, it argues the requirement of the energy model.

Understand the power dissipation is the basic for the further research of power management. Only when we realize the power of the component in different state, we can make efficient strategies to save energy. In addition, we will be able to find the right direction of research that decrease the power dissipation of computer systems and improve the energy efficiency. Several works have been done in modeling and understanding the power dissipation of the computer system. [76] develops an automated tool and use it to get the energy usage of various resource components. Masehri et. al. run a group of different benchmarks and uses subtract method to find out the power dissipation of the main computer components in [90]. In addition, [35] builds the energy model for the main components, which includes CPU, memory, disk and network interface card (NIC); using these model they compute the power dissipation of the components dynamically. Different with these works, in this chapter we use a method that directly measure the current on the wires of the ATX power connector. Our method can get more accurate power dissipation result for these components. Besides, our work can be a validation basis for most of the energy modeling work.

Before the year of 2000, people already realized that performance is not the only requirement when design the computer systems. Moore's law tells us that the trend of hardware improvement. Also, this trend implies the quickly increasing of power dissipation of computer systems. Then people inclined to design multi-core processor other than continually increase the frequency of the processor. Especially these years, a lot of work have been done to save the energy consumed by computer systems. The trend of power management is helpful for the future research work. In this chapter, we derive out some trends of power management by

doing a bunch of experiments.

Nowadays, nearly each computer component could work in several states, this makes the operating system could make out different power saving strategies based on the workload of the system and the user's usage habit. In [20], they add a dc-dc switching regulator to transfer the voltage into several smaller voltage to supply electricity for CPU's sub-units, in this way CPU could work in different power mode. New DRAM refresh control techniques [126] are also proposed to reduce the static power of memory. Also, some people argues that traditional DRAM should be replaced by new memory, such as PCM [82], in the future. The memory may work on different modes based on the accessing frequency of the memory. This makes the operating system or other programs can make different strategies to save energy.

3.3 Power Measurement

To understand the fine-grain power dissipation in a computer system and the trend of power management, we need to measure the power of the main components of the computer system in detail. This section describes how do we measure the power of CPU, memory, disk and NIC. Because of the power supply circuits on the motherboard of the computer is very complicated, we use specific method to measure the power of each component. In addition, we run some benchmark programs to generate different kinds of system usage and measure the dynamic power of the components. Although the description of our power measurement method is based on our experimental platform that we will address in the next section, it could be easily used on other platforms.

3.3.1 Power Measurement Problems

Some components, such as disk and CD-ROM, use a separate ATX power connector to supply electricity. We can measure the current on the cables of these ATX power connectors directly, then we can compute the power of these components with the measured result. But, for other components which usually connected with the motherboard, such as memory, CPU and NIC, we can not get the power of them by measuring the current directly. These compo-

nents use motherboard to supply electricity and it is difficult to understand the circuit of power supply module of the motherboard. The motherboard usually use a 20 pin or 24 pin ATX power connector to supply electricity; the voltage of cables with different color on ATX power connector is different. Figure 3.1 shows the 20 pin ATX connector of PC_{05}, while Figure 3.2 shows the 4 pin ATX connector of PC_{05}. All these two ATX connector are connected with the motherboard of PC_{05}. These two ATX connector supply electricity for the motherboard, then the motherboard supply electricity for other components that are plugged on the motherboard. Figure 3.3 is the 24 pin ATX connector of PC_{10}. In addition, PC_{10} also has a 4 pin ATX power connector as that of PC_{05}. In this way, components that work on different voltage get the voltage they needed from the motherboard.

Sometimes, second or third times of voltage conversion are needed for some components that ATX power connector do not directly supply the same voltage as their working voltage. In addition, some components may work on the same voltage. This means that the same color cables of the ATX connector may not only supply electricity for only one component. So, we can not directly get the power of these components by measuring the current of cables which supply the same voltage as the component. Finally, the power supply specification may different for different computer platforms. For example, the new computer system usually set a 4 pin ATX power connector to supply electricity for CPU separately, but some old computer system does not. So, it is very hard to measure the power for these components accurately, and the power measurement method should be related to the experiment platform. Basically, we use two methods, direct measurement and indirect measurement, to get the power of these main components.

3.3.2 Direct Power Measurement

We can get the power of some components by measuring the current of the ATX power connector of them directly. The old machines usually use the 4 pin peripheral power cable to supply electricity for this type of components. The voltage of the yellow and red wires on this

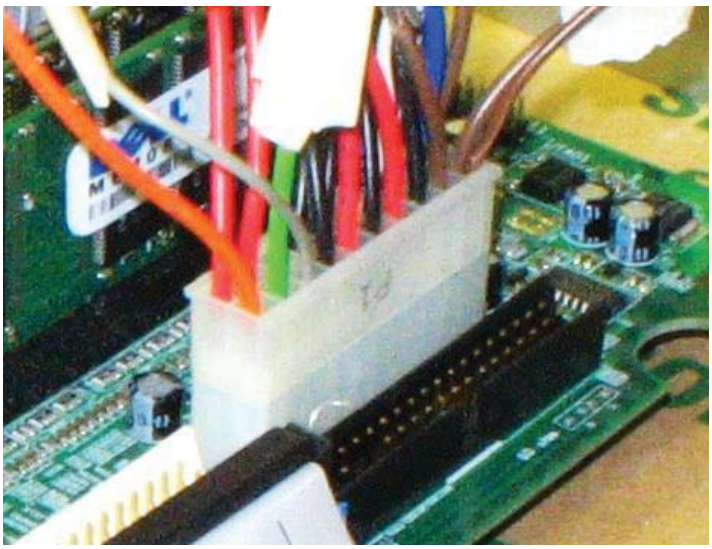


Figure 3.1: 20 pin ATX power connector of PC_{05}.

ATX power connector are +5V and +12V; the black wires are connected to the ground. The component may use one of these two voltages at different states. For example, when the disk is spinning up and down, it uses +12V voltage. The new machines usually use SATA power cable, which adds +3.3V voltage to the old 4 pin peripheral power cable (some SATA power cable may not have +3.3V wire). To measure the power of these components, we cut the wires except for the ground wires and connect multimeter to measure the current on the wires. The power dissipated on the multimeter is very small, so we can neglect it. Suppose, the current of a wire with voltage V_i is C_i . So, The power of a component of this type is :

$$P_d = \sum_{i=1}^{n} C_i \times V_i \qquad i = 1,2 \quad or \quad i = 1,2,3 \tag{3.1}$$

3.3.3 Indirect Power Measurement

Benchmark	Yellow-1	Yellow-2	Brown-1	Brown-2	Brown-3	Blue	White	Red-1	Red-2	Gray	Green	Yellow-0	Red-3	Red-4
MEM	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N
INT	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N

Table 3.1: Power supply relationship of components and cables of PC_{05}.

In order to measure the power of the components that supply electricity by the motherboard,

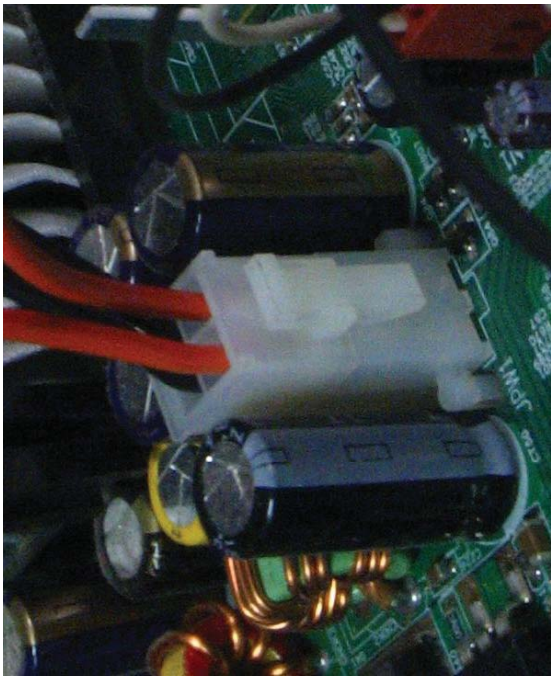


Figure 3.2: 4 pin ATX power connector of PC_{05}.

first we need to distinguish which cables of the ATX power connector that are related with this component. In addition, we need to find out whether these cables are supply electricity for this component only. To find this we need to measure the current on each cables except for the ground cables, then run different micro-benchmarks and see which cable's current changed after start the micro-benchmark. We use two simple micro-benchmarks to find out which cables are related with memory and CPU's power supply. IntegerTest (INT) benchmark, shown in Listing 1, is a simple program, which executes integer computation continually. It is a light weight benchmark that can fit in L1 cache, so that it will not incur memory reference after it is loaded. MemoryTest (MEM) benchmark, shown in Listing 2, is a program that access memory continually. When running this benchmark both the CPU and memory will be active. By setting different ARRAY_SIZE, MEM benchmark can also be used as L1CacheTest (L1) and L2CacheTest (L2) benchmark. Table 3.1 shows the relationship between cables and components on PC_{05}. This computer uses both 4 pin ATX power connector and 20 pin ATX power connector to supply electricity for the motherboard. Yellow-1 and Yellow-2 are cables

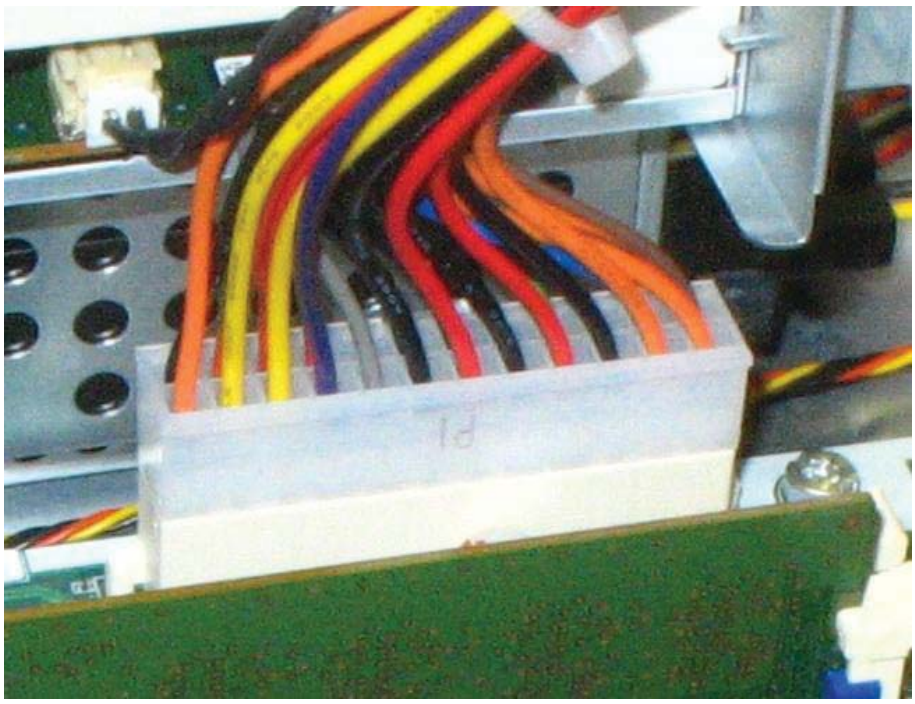


Figure 3.3: 24 pin ATX power connector of PC_{10}.

of the 4 pin ATX power connector, the others are cables of the 20 pin ATX power connector. After we know the relationship we can use different method to find out whether some cables supply electricity for a component only. The following paragraph will talk about how we get the power of CPU, memory and NIC in detail.

```
1 int main(int argc, char *argv[])
2 {
3     unsigned int a , b;
4     a = 1; b = 2;
5     while(1){
6        a = a+b;
7     }
8     return EXIT\_SUCCESS;
9 }
```

Listing 3.1: The code of integer benchmark.

```
1 #define CACHELINE_SIZE 64
2 #define L2CACHE_SIZE 2048
3 #define ARRAY_SIZE (L2CACHE_SIZE * 1024
4         / CACHELINE_SIZE * 2)
5 typedef struct{
6         int data[CACHELINE_SIZE/4];
7         }LINE;
8 LINE array[ARRAY_SIZE];
9
10 int main(int argc, char *argv[])
11 {
12     unsigned int size = ARRAY_SIZE;
13     unsigned int i = 0;
14     while(1){
15           array[i%size].data[0] = i;
16           i++;
17      }
18     return EXIT_SUCCESS;
19 }
```

Listing 3.2: The code of cache benchmark.

CPU's power

From Table 3.1, we know that only Yellow-1 and Yellow-2 supply electricity for CPU. In fact, most new motherboard use a specific 4 pin or 8 pin +12v voltage cable supply electricity for CPU, so that the CPU can get a very stable voltage. The two computer we used both use a 4 pin +12v voltage cable. We cut the cables except for the ground cables of the 4 pin ATX power connector and connect multi-meters to measure the current of these cables. The power

of CPU can be computed use Equation 3.2:

$$P_{CPU} = \sum_{i=1}^{n} C_i \times 12.00 \quad i = 1,2 \quad or \quad i = 1,2,3,4 \tag{3.2}$$

Our experiment result is very close to the specification of the CPU both at idle and busy state. Although part of power is dissipate by CPU power supply circuit, which is part of the motherboard, it makes sense to count it as power dissipated by CPU.

Memory Power

Also, we need to measure memory's power use indirect method. From Table 3.1 we can see that when we run the MEM benchmark, the current of the yellow wires of the 4 pin +12v voltage cable and the brown cables of the 20 pin motherboard power cable changed. The yellow wires of the 4 pin +12v voltage cable change because of the CPU, so the 3 brown cables of the 20 pin motherboard power cable are related with the memory. From Table 3.2 we can see that the power supplied by this three line is higher than the memory's specification power. This means other parts of component also use these three cables to supply electricity.

Brown-1	Brown-2	Brown-3	Power
1.418A	0.864A	0.92A	10.56W

Table 3.2: Brown cables measure result of PC_{05} when the system is idle.

For the purpose of getting the memory's static power, we need to set different memory number and measure the current. This platform uses two same size memories, we call them M_1 and M_2. First, we use two memories and measure the current to get a power P_1 use Equation 3.3. Then, we remove M_2 and measure again, we got P_2. Finally, we remove M_1 and change it to M_2 and measure it once more, we got P_3. With this three value we can get the total memory's power with Equation 3.4:

$$P_{SUM} = \sum_{i=1}^{n} C_i \times 3.3 \quad i = 1,2,3 \tag{3.3}$$

$$P_{Memory} = 2 \times P_1 - P_2 - P_3 \tag{3.4}$$

Strategy	Brown-1	Brown-2	Brown-3	Power
Both	1.418A	0.864A	0.92A	10.56W
M_1	1.117A	0.827A	0.73A	8.824W
M_2	1.166A	0.724A	0.77A	8.778W

Table 3.3: Brown cables measure result of PC_{05} when use different memory.

From the result, shown in Table 3.3, we get the power of the memory is 3.518W, this is close to the memory's specification power.

NIC's Power

Network interface card usually connected with the motherboard through the peripheral component interconnect (PCI) or peripheral component interconnect express (PCI-E) port, it is powered by the motherboard as the memory. We can measure the memory's power use indirect method, but it is hard for us to measure NIC's power. The reason is when the NIC is active memory and CPU are also active. So, we can not use the method we related before to distinguish which lines of the ATX power connector supply electricity for NIC. We can get the total power of the system when plug in the NIC and remove the NIC. The result shows that the total power of the system does not change on these two circumstances. This means that the power of NIC is very low, that we can nearly ignore it.

3.4 Experiments & Evaluation

Using the method described in the last section, we do several experiments on our two platforms. In this section, we will describe our experimental platform first. Then we will evaluate our experiment result from three angles, which include fine-grain power dissipation, energy model and the trend of power management.

3.4.1 Experiment Platform

Our experiment platform includes two desktop PCs, a new HP PC(PC_{10}) and an old Compaq PC(PC_{05}), which was bought about five years ago. Since these two PCs are produced by the same company, it is reasonable to use them to make an comparison. PC_{05} uses an Intel Pentium 4 2.0GHz uni-core processor. This CPU do not support DFS, it can only work on 2.0GHz. In addition, it has two 512MB DDR memories and an 80GB Seagate disk. PC_{10} uses an Intel Pentium E5300 2.60GHz dual-core processor, which support DFS and could work on more than eight frequencies. Also, the core could work on different voltages range from +1.110V to +1.328V. In addition, it has two DDR3 memories, one is 1Gb and another one is 2Gb, and has a 640Gb WD disk. Finally, all of these two platforms have no other components except the motherboard, the power supply, the CPU, the memory, the disk and the fans. We only concern on these regularly used components in this chapter. Table 3.4 shows the configuration of this two platform in detail.

Component	PC_{05}	PC_{10}
CPU	HP Compaq Intel Pentium 4 2.0GHzv 1 core Core Voltage 1.471V 512KB L2 Cache 8KB L1 Cache	HP Intel Pentium E5300 2 cores Core Voltage 1.100V 2048KB L2 Cache 32KB $\times$ 2 L1 Cache
Memory	DDR 256MB $\times$ 2 Frequency 132.9MHz Cycle Time 6 clocks	DDR3 1GB + 2GB Frequency 399.0MHz Cycle Time 15 clocks
Disk	80GB Seagate Disk	640GB WD Disk

Table 3.4: Experiment platform configuration.

Except using multimeter to measure the power of a specific component, we also use a ”Watts Up” to measure the power of the system. In this chapter, we only concern on several main components of the main frame, so the ”Watts Up” is connected with the main frame’s power cable. The monitor uses a separate power cable to supply electricity.

Finally, we use five micro-benchmarks, the INT benchmark, MEM benchmark, L2 bench-

mark, L1 benchmark and Prime95 (PRIME) benchmark, to generate the dynamic power. Listing 1 is the code of INT benchmark. Use the code of Listing 2, when we set different ARRAY_SIZE and CACHELINE_SIZE, it becomes MEM benchmark, L2 benchmark and L1 benchmark. The difference of these three benchmarks is that the data will be read from 3 different storage levels in the cycle. PRIME benchmark is said to be the most severe benchmarks of CPU. It could make the CPU nearly dissipate the highest power. All these five benchmarks can make the percentage of CPU utilization come to 100% and generate stable power dissipation. So, we can use these benchmarks to find out how much dynamic power the components of the computer system dissipate.

3.4.2 Fine-grain Power Dissipation of PC

The first aim of this chapter is to tell the fine-grain power dissipation in a computer system. From Figure 3.4 we can see that the static power of the new CPU has decreased about 45 percent compared with the old one. Also, it shows that even though the size of memory has increased 2 times compared with the old one, the static power of memory decreased about 64 percent. Finally, we can see that the static power of disk of these two platforms are about the same.

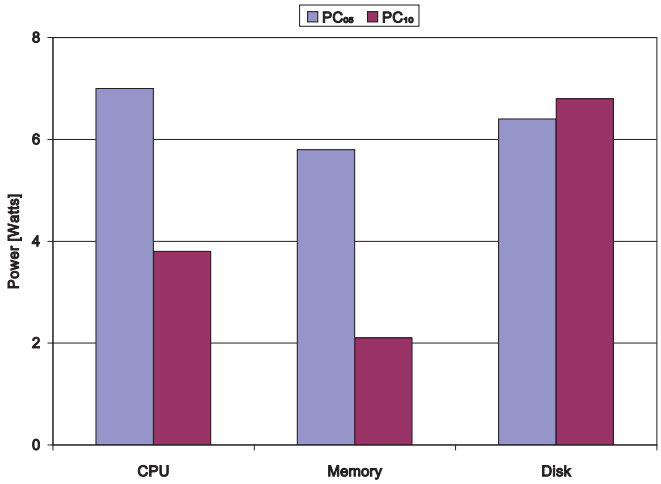


Figure 3.4: Power dissipation comparison of the main component.

From Figure 3.5, we can see that on PC_{05} the CPU's power dissipation accounts for 15 per-

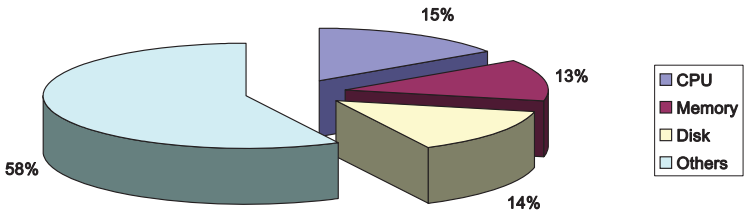


Figure 3.5: Pie chart of the power dissipation of PC_{05}.

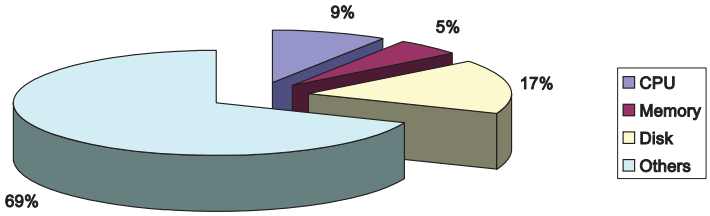


Figure 3.6: Pie chart of the power dissipation of PC_{10}.

cent, which is only slightly higher than memory and disk. All the other parts of PC_{05} account for about 58 percent, which is mainly dissipated by the power supply and the motherboard. This two components, as we can see, dissipate more than half of the total power of the computer system on PC_{05}. Although we can not measure their power directly, if we assume the transfer efficiency of the power supply is 85 percent, then the power supply dissipates about 6.8W and the motherboard dissipates about 19.3W.

Figure 3.6 shows that the power dissipation of disk of PC_{10} accounts for about 17 percent, which is much higher than the power of CPU and memory. As we know from Figure 3.4, the power of disk does not change a lot between our two experiment platforms. The static power of CPU is not the dominant anymore in these three components when the system is idle. In addition, we can see that the other parts accounts for 67 percent, which is higher than the PC_{05}. Also, we assume the transfer efficiency of the power supply is 85 percent (the transfer efficiency of power supply does not improve in the last several years [125]), then the power of power supply is 6.15W and and power of motherboard is about 22.1W. This result can generally tells us that the power dissipation of motherboard and power supply also do not decreased in the last several years.

Since we can not set the system working on a specific percentage of CPU, memory or disk utilization in a time interval except fully utilized or fully idle. We compare the dynamic power of these devices when they are fully used. We run a bunch of benchmarks to make these devices busy, then we measure the power of these devices. From Figure 3.7 we can see that the dynamic power of CPU is different when running different benchmarks. Moreover, this figure shows that the dynamic power of PC_{05} is higher than PC_{10} when run each benchmark. Especially, when running the MEM and INT benchmarks the dynamic power of the new CPU decrease more than 35 percent compared with the old one. This means that the INT computing, bus control and BUP sub-unit's dynamic power dissipation have decreased significantly.

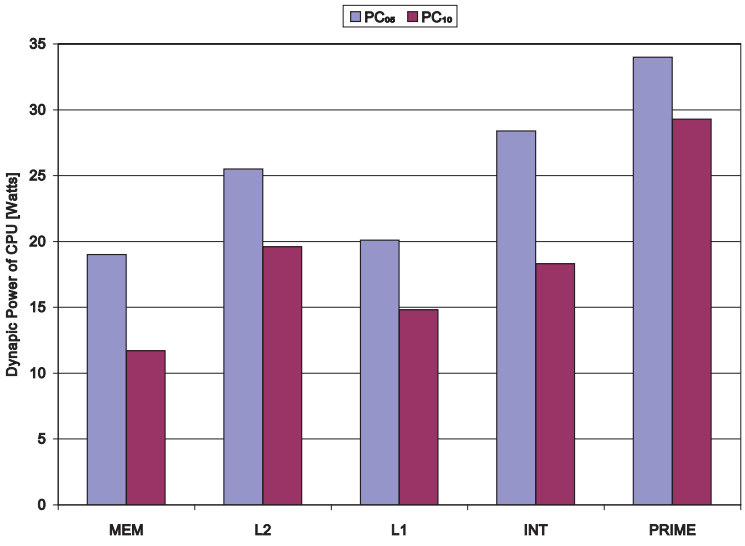


Figure 3.7: Dynamic power of CPU.

We use three different benchmarks, MEM, L2, L1 benchmarks, to generate the dynamic CPU usage. The difference of these three benchmarks is that it read data from three different storage levels. L1 benchmark only reads data from L1 cache, and the other two benchmarks read data from both L1 and L2 cache. The access frequency of these three benchmarks is different because of the read latency of these three level of storage is different. Figure 3.8 shows the percent of cache's power relative to the CPU's power. We can see that it accounts for more than 70 percent for both platforms when run each benchmark. In addition, on PC_{10}

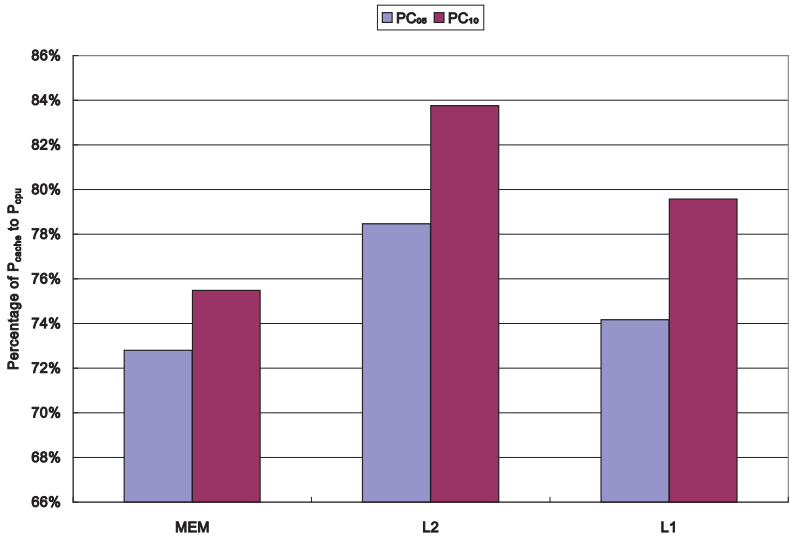


Figure 3.8: The impact of the power of cache on the power of CPU.

when run L2 benchmark it is more than 80 percent. Also, from Figure 3.9 we can see that cache's power accounts for nearly more than 20 percent for both platforms when run these three benchmarks. When run L2 benchmark on PC_{05}, it nearly hit 35 percent. This means that cache accounts for a large amount of the power dissipation.

3.4.3 Energy Model

Energy model is the key for power profiling with software. Only with the accurate energy model we can give out the power dissipation of each component on system level or even process level. Another important contribution of our work is that, from the experiment result we find the factors that are related with the components. With these observation we come out the energy model for CPU and network.

Energy Model of CPU

We know that the power dissipation of CPU includes two parts the static power (P_s) and the dynamic power (P_d). The static power of CPU does not change with the workload or frequency of CPU. Figure 3.10 shows that when the system is idle, the static power of the CPU does not change as we change the frequency of the CPU. Also, the total power of the system does not

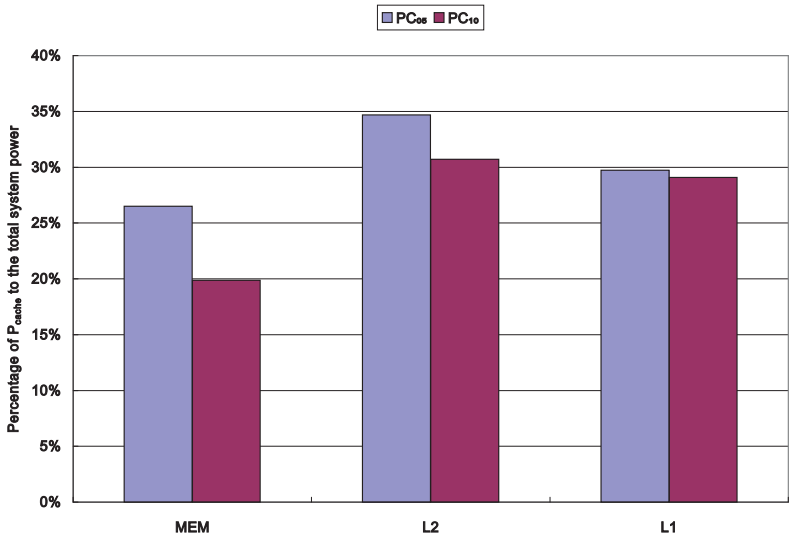


Figure 3.9: The impact of the power of cache on the power of the whole computer system.

change. When we running different benchmarks under different CPU frequencies, the CPU's power increases gradually with the increasing of CPU frequency. Figure 3.11 shows this result. In addition, Figure 3.11 shows that the total power of the system also increased gradually as the CPU's power. The reason is that these two benchmarks we used to do this experiment only generate CPU power dissipation, so the increase of CPU's power is the increase of the power of the system on each step. In addition, Figure 3.12 shows the percentage of CPU's power relative to the system when run INT and L2 benchmarks with different frequencies. We can see that the dynamic power of CPU has linear relationship with the frequency of CPU. From our result, we come to an energy model for CPU as shown of Equation 3.5.

$$P_{CPU} = P_s + k \times F \times (\delta_1 P_1 + ... + \delta_n P_n) \tag{3.5}$$

In this equation, F denotes the current CPU frequency. P_1, ..., P_n denotes the power of each sub-units of CPU and δ_1, ..., δ_n denotes the current percentage of utilization of each sub-units.

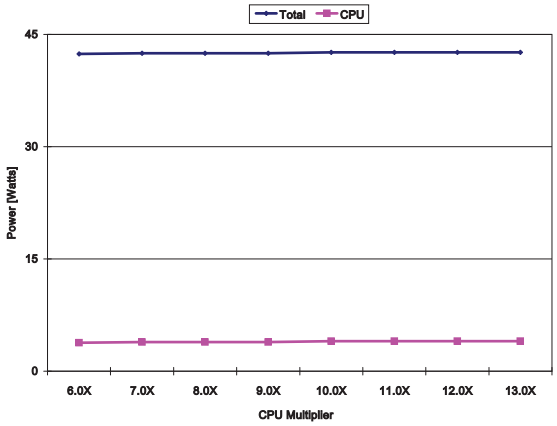


Figure 3.10: The static power of the system and the CPU with different CPU frequencies on PC_{10}.

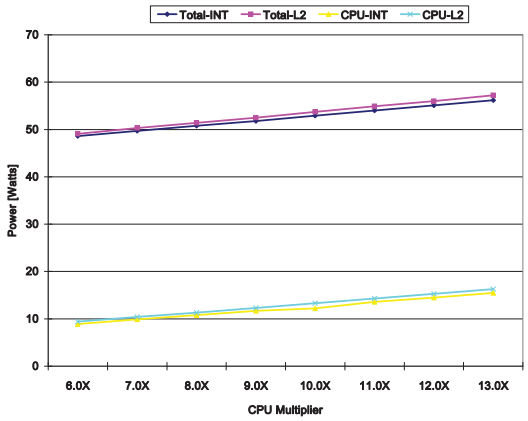


Figure 3.11: The power of the system and the CPU with different CPU frequencies and running INT and L2 benchmarks on PC_{10}.

Energy Model of Network

In addition, it is also useful to know the energy model for an application that we usually used, for example network downloading. When we downloading files with our experimental platforms, we find that the total system's power has linear relationship with the downloading bandwidth. In addition, when we downloading a file, we find that CPU, Memory and Disk are all active. Although, the NIC is also active, but our experiment shows that NIC nearly has no

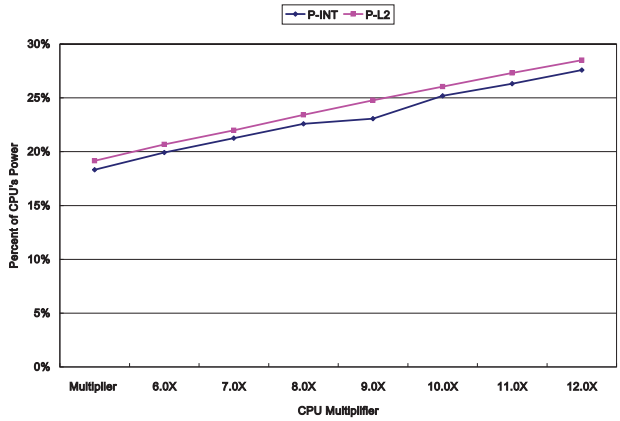


Figure 3.12: The percentage of CPU's power when running MEM and L2 benchmarks.

power dissipation. So, we can come out the energy model for network downloading:

$$P_{NET} = k \times B \times (\delta_1 P_{CPU} + \delta_2 P_{MEM} + \delta_3 P_{DISK}) \tag{3.6}$$

In Equation 3.6, B denotes the downloading bandwidth.

3.4.4 The Trend of Power Management

From our experiment result we can generally find that, in the last several years, power management of computer systems grabs more and more concern by academic and production areas. System performance is not the only consideration any more when design a new computer system. But, most components of the computer system, such as disk, power supply and motherboard do not dissipates less power than before.

The Power Management of CPU

From our experiment we observe that the static power of CPU has decreased dramatically in the last several years. This means that CPU has used more and more power saving techniques, like clock gating and DVFS. The clock gating technique is used on more sub-units in new CPU thus the static power of new CPU have decrease greatly. In addition, DVS and DFS technique is efficient method to decrease the waste of energy when the system is in idle state.

The performance is not the only consideration, people began more and more concern on power management.

In addition, we find that the dynamic power of CPU does not decrease. This shows that the power of the sub-units of CPU do not decreased because the CPU use more and more transistor and the density of CPU increased quickly. Therefore, a lot of work need to do to decrease the CPU's dynamic power through the IC design level. Our experiment shows that the dynamic power of CPU may accounts for more than 70 percent of the system, so it is valuable to work on this area to save energy.

The Power Management of Memory

The static power of memory are closely related with the size of the memory, because most power is dissipated on the refreshing of memory. From our experimental platform we can see that the size of the memory has increased 3 times, but the static power decreased about 64 percent. This means in the last few years the power management of memory has acquired great success. The density of memory is much higher than before, however it dissipates much less power. Also, new refresh techniques are proposed to make the memory refresh operation executed in an energy saving mode. Although the static power of memory is only about 0.7w per Gb for a general computer system, the memory's power dissipation occupies the leading position than other devices for some large computer system, such as the server in the data center. A new trend of memory energy saving is replace traditional DRAM with non-volatile storage materials, such as PCM [82]. From our observation, we know that the power dissipation of memory is much less than other components on PC_{10}, so memory is not a big deal in the future several years.

The Power Management of Disk

Our experiment shows that the static power of the disk does not change in the last few years, although new techniques have been used on disk, which add more work state to the disk. This enables the operating system make out power saving strategies. Our experiment shows that the power dissipation of the disk even increased in the last few years. In addition, we find the percentage of disk's power dissipation increased in the last few years.

The Power Efficiency of Power Supply

Although the 80 plus program [125], in which most computer producer agree that the future produced power supply should be come to an transfer efficiency of more than 80 percent, was agree by most producer of computer system. In [18], researchers argue that that the transfer efficiency of power supply does not improved in the last few years. Since, all the energy consumed by power supply is not used for computing and it accounts for 15 percent of the system's total power, so it is a good topic to work on this area.

The Power of Motherboard

Finally, we observed that the power dissipation of the motherboard did not decrease in the last several years. In addition, the power of motherboard accounts for about 20 percent of the system. So, a lot of work still need be done on the circuit design of the motherboard. The development of low power motherboard is significant for the decreasing of the system's power.

3.5 Implications

After these observations, we are now in a position to derive several important implications for future energy efficient system design, especially we give a few common but wrong assumptions made by previous work in the field.

3.5.1 CPU Utilization

Software power profiling is an critical step of power efficient system design. A bunch of work [76, 90, 35] build their energy model based on CPU utilization. However, our experiment shows that CPU utilization is not strictly related with CPU's power. In Figure 3.7, all of these five benchmarks can make the percentage of CPU utilization come to 100%, but we can see that the power of the CPU is different for both platforms. The PRIME benchmark, which is the most severe CPU benchmark, makes the CPU come to a highest CPU power dissipation, while the MEM benchmark generates the least power dissipation. The difference of power dissipation of CPU is mainly because of the using of clock gating technique. These benchmarks use different sub-units of CPU, which have different power dissipation. Canturk Isci et. al. proposed a more accurate CPU power model in [67], this power model considers nearly all the sub-units of the CPU. But, this work is based on the CPU of the same period as PC_{05}. In addition, using this model we need a fully understanding of how the CPU works. So, it is hard to be used on the new CPU directly. We find out that conventional definition of cpu utilization is not a good indicator of power dissipation, thus, previous cpu utilization-based power management schemes need to be revisited. Finding a good indicator of power dissipation is an urgent and open problem.

3.5.2 Controllable Cache Size

In order to improve the system performance, the cache size increased quickly. But, we observe that the power of cache sub-unit account for more than 70% of the CPU's total power dissipation. In Figure 3.7, we can see that L2 benchmark makes CPU dissipates more power than MEM benchmark. Because of the frequency difference of memory and CPU, the CPU have to add several idle time periods to wait for the memory. This means the workload of L2 benchmark is much higher than MEM benchmark, so it dissipates higher power than MEM benchmark. In addition, we find that L1 benchmark dissipates less power than L2 benchmark. That's because the size of L2 cache is about 4 times of L1 cache. The power dissipation of

cache sub-unit is directly related with the cache size. Our result is the same with the result in [67]. Since performance of processor is not the only consideration and cache consumes a lot of energy, there must be a tradeoff between performance and power dissipation. The cache size should consider the realistic usage. Sometimes the lower performance is enough for the tasks, for example when watching a low resolution video. For scientific software, performance is the most important thing. So a controllable cache size could certify both the performance and power saving requirements. The operating system should be able to decide the used cache size based on the realistic requirements in the future.

3.5.3 Higher Transfer Efficiency are Needed

Because many components of the computer system work on different voltages, the power supply must transfer the AC to different voltage of DC. In addition, the voltage needs to be transferred another one or several times to meet the specific needs of a device. These voltage transfer circuit may on the motherboard or in a device. For example, the CPU use a 4 pin ATX power connector to supply +12v voltage, the work voltage of the new CPU's core is only about +1.1v. Thus the +12v voltage needs to be converted to the real work voltage of the core by the voltage regulator module (VRM). Ideally, the transfer efficiency is 100 percent, but in realistic the transfer efficiency is less than this value. For example, the transfer efficiency of power supply is about 85 percent for these new power supplies. Much energy has been wasted during this process. We use fans to make an experiment because we can measure the power of fans directly. First, we use multimeter to measure the current on the electricity cable on the CPU fan and computer case fan. Then, we read the total system power before and after we remove these two fans. Figure 3.13 shows that, the transfer efficiency of fans is only about 25 percent. 75 percent of energy that used to supply electricity for fans is wasted. This part of energy can be saved if the voltage of fans convert less times and the transfer efficiency of power supply is improved. The future design requires that more components work on the same voltage, thus the transfer times of voltage could be decreased. Also, it is valuable to work on the voltage

transfer area.

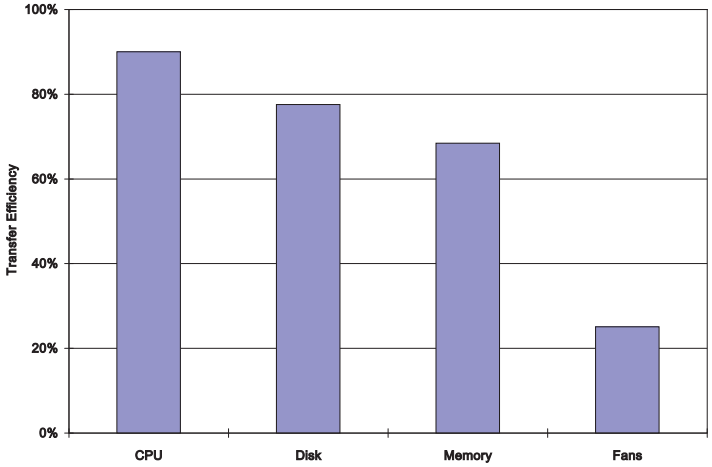


Figure 3.13: Transfer efficiency of CPU, memory, disk and fans.

3.5.4 Multi-core Task Allocation

With the increasing of the cores in CPU, how energy-efficiently schedule the tasks on cores becomes a critical problem. Some work [80, 9] have already proposed method about how to efficiently schedule jobs on multi-cores so as to save energy. One method is make part of the cores work in lower workload or idle state while others in higher workload; another method is make all the cores as busy as possible, after the task is finished then make all of them in idle state. We made an experiment to verify which method is more energy-efficient.

PC_{10} uses a two core processor. First, we running each benchmark to make both cores busy and record the CPU's power dissipation. Then we run each benchmark and designate it running on the first core then record the CPU's power dissipation. Finally, we run each benchmark and designate it running on the second core then record the record. The last two test will make only one core of the CPU in fully busy state and another CPU in idle state. Figure 3.13 shows that when set one core idle and one core busy the CPU's power dissipation is only slightly less than when the two cores of CPU is all busy. When we run the MEM benchmark to make all the cores busy, it even generate a less CPU power dissipation than the other two circumstances. The result shows that the first method does not really save energy. The second method is a

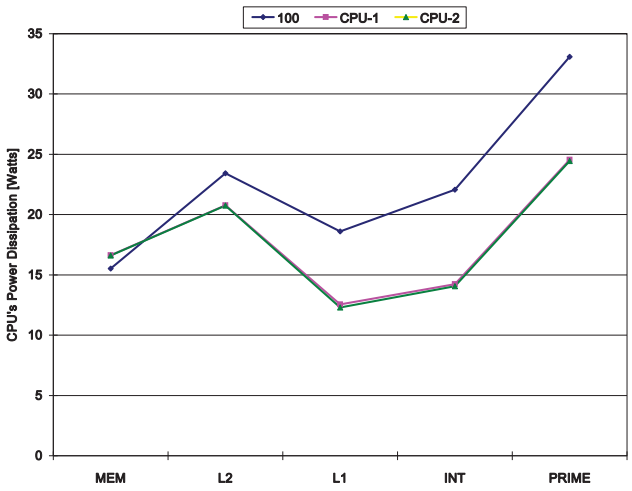


Figure 3.14: The power of CPU when run benchmarks on different cores.

good choice since it could make the CPU work longer in low power dissipation state.

3.6 Summary

In this chapter, we use two experiment platforms of different period (PC_{05} and PC_{10}) to measure the power dissipation of several main components of these two computer systems. From our experiment result, we give a clearly view of power dissipation in a computer system. In addition, we derive out some trends of power management and some implications that are helpful for energy efficient system design.

First, we describe our experimental method in detail based on our experiment platform. Our method can be easily used to measure the power dissipation of other platforms. More importantly, we introduce a method to give out the power supply relationship between cables of the ATX power connector and the components of the computer system. It is useful for the future even if the specification of the ATX power supply connector changes in the future. In addition, we introduce the direct and indirect power dissipation measurement method, which could be used to measure nearly all the main components of the computer system.

Using our power measurement method, we measure the power dissipation of the main components of the computer system. The result shows that the static power of CPU and memory

have decreased a lot in the last several years. While, the power dissipation of disk, power supply and the motherboard do not change too much. In addition, we find that the total power dissipation of the new platform is less than the old platform. Based on the observations and the analysis of the result, we derive out some trends of power management of computer systems in the last several years.

Finally, based on our observations we derive out several critical implications that are helpful for energy efficient system design, shown in Table 3.5. Our result shows that CPU utilization can not reflect CPU's power accurately. Because when run different benchmarks, all of which make the CPU fully used, the power dissipation of the CPU is greatly different. In addition, our experiment result shows L1 and L2 cache account for a large amount of the CPU's power dissipation. Thus, if the cache size is controllable by the operating system, we can make out efficient energy saving strategies by decrease the size of the cache. Also, some component's transfer efficiency is much lower than we thought before, the voltage transfer times should be decrease in the future. Finally, we find it is an mistaken idea that scheduling the tasks on the multi-core system to make part of cores idle and another part busy can save energy.

By conducting measurement and research in this chapter, we achieved our first objective. Particulary, we identified that CPU dynamic power still dominates as much as 70% of the whole system power dissipation although the CPU static power has been reduced significantly over the past five years. Dynamic power dissipation quantifies the resource usage in a computer system at a certain level, which is closely related to software workload activities. In order to better understand the behavior of workload in association with the power dissipation, especially CPU dynamic power, hardware instrumentation is an necessary approach. However, due to the complication and cost of the equipment, this hardware approach is not always available. In addition, there are circumstances that all we need to analysis is not the particular power values but the general shape of the power dissipation. Run-time estimation techniques fits these needs well. In the next chapter, we will analyze the CPU dynamic power dissipation and attempt

to identify a power model that estimate power dissipation of a CPU within an accepted error range.

Target	Observations	Implications
CPU	When the percentage of CPU utilization is 100%, the power is different when run different benchmarks.	CPU utilization is not a good indicator of the power dissipation of CPU.
	CPU frequency has linear relationship with the dynamic power.	Frequency control is efficient method to control the power and temperature of CPU.
	Cache dissipates a large amount of power when it is used. The power dissipation of cache is related with cache size.	Cache size should be controllable in the future, so that the operating system can tradeoff between performance and energy saving requirements.
	In a multicore system, idling part of cores while keeping the rest of cores busy does not decrease the total power of CPU too much.	The task allocating strategy that make part of cores idle while the other parts busy is not the efficient strategy that save energy.
Memory	The power of memory of PC_{10} is much less than that of PC_{05} although the memory size of PC_{10} is 3 times of PC_{05}'s.	In the near future, memory power should not be a big problem on desktop machines.
Disk	The power of disk of PC_{10} is about the same as PC_{05}'s.	The disk power dissipation is stable and as the inception of solid state drives, the disk power should not be a major problem too.
Power Supply	The more times the voltage of a component are transferred the less the transfer efficiency of this component is.	The voltage transfer times of the components should be decreased. All the components of the computer system should work on the same voltage.
	Power supply and motherboard dissipates more than half of the total power of the system.	The transfer efficiency of power supply need to be improved and it is valuable to research on this area.

Table 3.5: A Summary of Observations and implications.

CHAPTER 4

SPAN

In the previous chapter, we observed the trend of power dissipation design in computer system. Undoubtedly, the dynamic power dissipation of the major component in the system occupies a large portion of the whole system. This part of power dissipation is triggered by system resources usage, such as various registers, caches, pipelines in a CPU. In this chapter, we take a insight analysis of what is the major indicator of CPU dynamic power dissipation. If such indicator exists, we are able to approximate the power dissipation of a CPU without hardware instrumentation. In this chapter we mainly introduce how we model the run time power dissipation of a CPU. Using the model, we propose SPAN, a set of APIs for function level power profiling.

4.1 Introduction

Understanding the power dissipation behavior of an application/workload is the key to designing energy-efficient computer systems. Power modeling based on performance monitoring counters (PMCs) is an effective approach to analyze and quantify power dissipation behaviors on a real computer system. One of the potential benefits is that software developers are able to optimize the power behavior of an application by adjusting its source code implementations. However, it is challenging to directly relate power dissipation to the execution of specific segments of source code. In addition, the existing power models need to be further investigated by reconsidering multicore architecture processors with on-chip shared resources. Therefore, we need to adjust PMC-based power models from the developers perspective, and reevaluate them on multicore computer systems.

In this chapter, we propose a two-level power model that estimates per-core power dissipation on chip multiprocessor (CMP) on-the-fly by using only one PMC and frequency infor-

mation from CPUs. The model attempts to satisfy the basic requirements from developer point of view: simplicity and applicability. Based on this model, we design and implement SPAN, a software power analyzer, to identify power behavior associated with source code. Given an application, SPAN is able to determine its power dissipation rate at the function-block level. We evaluate both the power model and SPAN on two general purpose multicore computer systems. The experimental results based on *SPEC*2008*C jvm* benchmark suite show the average error rate of 5.40% across one core to six core validation. We also verify SPAN using the FT benchmark from NAS parallel benchmark suite and a synthetic workload. The overall estimated error of SPAN is under 3.00%.

4.2 Two-Level Power Modeling

Actually, the power dissipation of a given platform can be divided into two parts:

- Baseline Power: the static power dissipation to maintain a system running. To be specific, static power of a motherboard, CPU, memory, CPU fans, and other components contributes to this part of the power dissipation.

- Dynamic Power: the power dissipation due to a task execution. By executing workloads on different platforms and different frequencies, dynamic power varies considerably. Other contributing factors could be temperatures, characteristics of workloads, and component utilizations.

The first primary goal of this chapter is to find a practical power estimation model describing dynamic power on multicore power-aware processors by using as few PMCs as possible. The essence of utilizing PMCs to estimate power dissipation is about information trade-off. The more PMCs information is retrieved, the more detailed and accurate the power model could be. However, collecting PMCs sometimes can be troublesome. First, commonly used processors cannot support retrieving more than a certain number of counters simultaneously. Previous models proposed [72, 28] necessitate multiplexing the counters so that several of them

can be accessed for one benchmark. Besides, types and names of the monitored events vary from platforms to platforms [49]. Usually, the power model established on one platform is not necessarily extensible. For example, Goel *et al.* differentiate PMCs in their power model for four platforms. Additionally, sampling PMCs usually means system overhead, which can be overwhelmed when the number of PMCs becomes large.

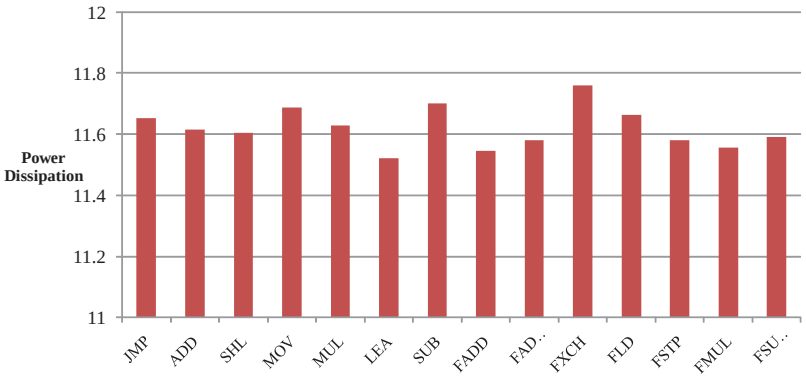
On the other hand, in order to describe the power characteristics of a given platform throughout, a fine-grained power model usually is trained by a large set of benchmarks. For example, Bertran *et al.* [12] develop approximate 97 benchmarks to exercise the power components on a single CPU. As a result, the training process could be unexpected long.

The production of this section is a set of power models with three basic features. First, the models have to provide acceptable high accuracy. Second, the parameters of the power models can be retrieved through a simple training procedure, which can be applied practically. In addition, the model input, the total number of PMCs, has to be maximally reduced to avoid multiplexing counters.

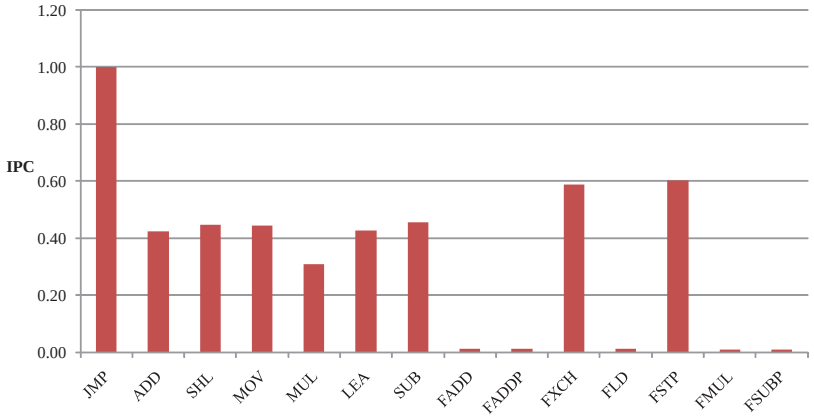
4.2.1 Observations

Leveraging PMCs, the most obvious method is to discover the possible correlation between a specific PMC and the power dissipation. The training benchmarks fulfill the task of PMCs selecting according to correlation coefficients. After obtaining training data, usually, researchers develop a linear regression model to derive a power model. Previous approaches concentrate on the mathematical methods to eliminate outliers, and to achieve high accuracy. However, few of them focus on direct factors influencing power dissipation, such as frequency.

One example is the argument on IPC. Indeed, an IPC value does reflect the power dissipation with high correlation coefficients for various workloads. However, the relationship between them can be weak under certain circumstances. We generate IPC ranging from 1 to 0.01 by continuously executing a single X86 instruction as Figure 4.1(a). The results of the corresponding CPU power dissipation are shown in Figure 4.1(b). The overall correlation



(a) Different instructions with their IPCs.



(b) Different instructions with their power dissipation.

Figure 4.1: Different instructions with their IPCs and power dissipation.

coefficient is 0.41 in this case. Nevertheless, the standard deviation of power dissipation is only about 0.067W for the given range of IPC between 1 to 0.01. Rarely could IPC make a representative factor of power dissipation in the similar scenarios.

However, other than those extreme benchmarks, regarding real applications, values of some PMCs reflect the power dissipation well. As Figure 4.2 illustrates, the IPC and the power dissipation present the correlation coefficient as high as 0.98 for the NAS parallel benchmark suite.

Given the above results, we observe that the same PMC possibly has changeable effects on the power dissipation prediction. One possible solution could be to profile different PMCs

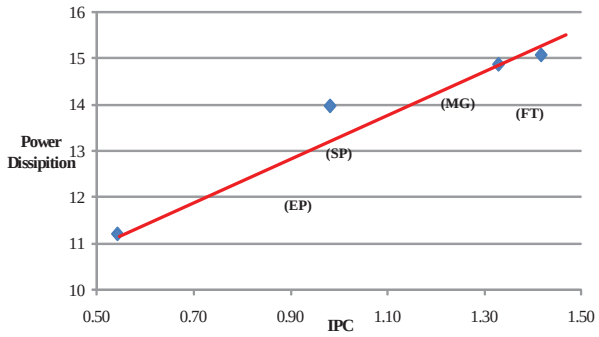


Figure 4.2: IPC profile and the power dissipation of NAS parallel benchmarks (The correlation coefficient is as high as 0.98).

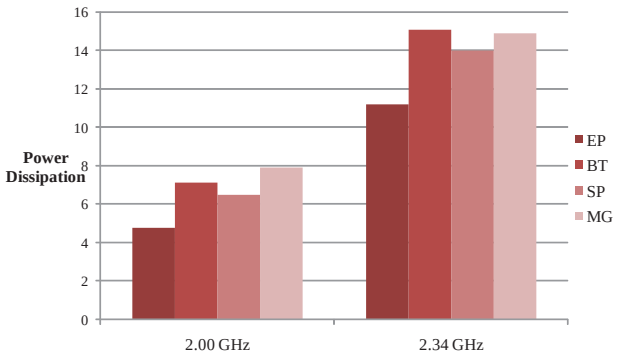


Figure 4.3: Power dissipation of NAS parallel benchmarks operating under two frequencies.

for each task and then select the most related ones, which is probably impractical in reality. In order to minimize the uncertain effects from PMCs on power estimation, we attempt to find an overall frame restricting the power estimation range. Inspired by [124] and based on our observation, the operating frequency fits the position well. Figure 4.3 shows the power behavior of four NAS parallel benchmarks executing under the frequency of 2.34GHz and 2.00GHz. Clearly, we are able to find the boundaries separating power dissipation, regardless of the types of the benchmarks, according to its operating frequency; thus, we establish a power model using frequency as the first level intuitively.

Generally speaking, we expect the power model to fully explore the possible relations between PMCs and power dissipation. Besides, if PMCs fail to provide positive information, the model will be able to minimize the disturbance introduced by it.

4.2.2 Methodology

In this section, we describe the methodology that produces the power models. In short, our approach follows the common modeling steps: defining the model input, generating microbenchmarks, training the power model, and applying the power model.

Considering inputs, the essential strategy is trade-off. On one hand, high accuracy necessitates more information from PMCs collected as inputs. On the other hand, the less PMCs we use, the more flexible and applicable the power model could be. We adopt only one PMCs to preserve the simplicity and to demonstrate the effectiveness of our two-level modeling. The PMC utilized in our study is IPC, as aforementioned. We design microbenchmarks carefully after selecting the model inputs. Totally, we test over 30 microbenchmarks stressing the CPU. By carefully reviewing, adjusting, and filtering them, we decide to choose 12 benchmarks for the training purpose because enough information can be provided from executing them. The training process is highly related to frequencies and IPC; however, we do not use linear directly, which most of the others do.

It should be noted that the main differences of our methodology from previous work are

three-fold: we incorporate frequency information in the power model, we use minimum size of PMCs, and the methodology can be applied to other platforms easily.

Basically, we deploy IPC along with frequency as the model inputs. Actually, a strong relationship between Instruction Per Cycle (IPC) and power dissipation is established in previous work [28, 84]. Although, in most cases, an IPC value is able to reflect the overall power dissipation, there are two issues by using IPC solely. First, different micro-operations might have various IPC values but similar power dissipation. For example, usually Floating Point Unit executes instructions much slower than Integer Arithmetic Unit nevertheless the power behaviors of them are similar. This problem can be easily solved if we consider each CPU component, such as FP, INT, and BPU (Brunch Prediction Unit) separately. In our case, it is not an option because we target on minimizing the PMCs in the model. Second, as the aforementioned, because power behaviors of a CPU are mainly limited by its operating frequencies, by using IPC, there is some marginal effect. As the IPC becomes large or small enough, the effects of IPC on power dissipation drop noticeably.

Our solution for the first issue is using IPC as a second level power indicator that tunes the estimation results obtained according to operating frequencies. In order to eliminate the marginal effect, we divide benchmarks into different categories based on the IPC values; then, we collect data and derive the model separately for each category. We demonstrate our approach as follows in detail.

Power Modeling

We denote the CPU frequency as F. Assuming that a CPU supports various frequencies, f_i, $i = 1,2,3...n$, we attempt to obtain the power dissipation information, $P(f_i)$, for each frequency f_i. Given a set of training benchmarks T with its sub benchmarks t_j, $j = 1,2,3...m$, executing under frequency f_i, we denote the power dissipation as $P(t_j, f_i)$ respectively. We calculate $P(f_i)$ as the median of $\{P(t_1, f_i), P(t_2, f_i), ..., P(t_m, f_i)\}$; thus $P(f_i)$ is resistant to outliers

statistically. Besides, we represent IPC of each benchmark as $IPC(t_j, f_i)$. Similarly, the median IPC value of all the training benchmarks is defined as $IPC(f_i)$. In most cases, the benchmarks with the median value of $P(t_j, f_i)$ also contribute the median value of $IPC(t_j, f_i)$. We describe $P(f_i)$ and $IPC(f_i)$ as *power pilot* for frequency f_i.

Second step, based on the *power pilot*, we compute $\Delta P(t_j, f_i)$ as the difference between $P(f_i)$ and $P(t_j, f_i)$ for each training benchmark. Similarly, we calculate $\Delta IPC(t_j, f_i)$ as the IPC difference of training benchmark t_i to the median value.

$$\Delta P(t_j, f_i) = P(t_j, f_i) - P(f_i) \tag{4.1}$$

$$\Delta IPC(t_j, f_i) = IPC(t_j, f_i) - IPC(f_i) \tag{4.2}$$

Targeting on predicting $\Delta P(t_j, f_i)$, we use $\Delta IPC(t_j, f_i)$ as model input to derive linear regression parameters, $P_{inct}(f_i)$ and $P_{\Delta}(f_i)$ as Equation 4.3 shows. The final predicted power dissipation is shown in Equation 4.4. We simply need to change $\Delta IPC(t_i, f_i)$ to be the actual $\Delta IPC(a_i, f_i)$ before applying the model to the i_{th} benchmark from task set $a_1, a_2, a_3, ...a_n$.

$$\Delta P(t_j, f_i)_{pret} = P_{inct}(f_i) + P_{\Delta}(f_i) * \Delta IPC(t_j, f_i) \tag{4.3}$$

$$P(t_j, f_i)_{pret} = \Delta P(t_j, f_i)_{pret} + P(f_i) \tag{4.4}$$

It is easy to notice that the most majority of power dissipation is determined by $P(f_i)$, which stems from frequency characteristics forced on each training set although the regression model is applied to $\Delta P(t_j, f_i)_{pret}$. Because $P_{inct}(f_i)$ and $P_{\Delta}(f_i)$ usually are small enough, we limit the inaccuracy from those power-irrelevant IPC values while reserve the positive relation between most IPC values and power dissipation.

As aforementioned, one shortcoming of using IPC solely is the low accuracy produced

when the values of IPC are either too high or too low. In order to constrict this marginal effect, we have to manipulate the given training benchmark set accordingly. First, we order the training set T with descending IPC, which yields $T_{ordered}$. Second, we divide $T_{ordered}$ into three categories with respect of their IPC values. Heuristic results, based on the average accuracy provided, show that the separating points locate approximately at 0.87 and 1.86. As a result, there are three groups of benchmarks: the one with relative low IPC, T_{low}, with average normal IPC, T_{normal}, and with relative high IPC, T_{high}. For each group, we apply the same method to obtain $P(t_{IPC\_level}, f_i)$, $IPC(t_{IPC\_level}, f_i)$, $P_{inct}(t_{IPC\_level}, f_i)$, and $P_{\Delta}(t_{IPC\_level}, f_i)$, where IPC_level represents low, $high$, and $normal$.

We use an accumulative approach for modeling multiple cores based on the assumption that each core has similar power behavior. Therefore, we apply the single core model to each core in the system. Specifically, we express the total power dissipation estimation as the follows:

$$P(a_j, f_i)_{pret\_total} = \sum_{k=1}^{k=cores} (\Delta P(a_j, f_i, k)_{pret} + P(f_i)) \tag{4.5}$$

where a_j is the target benchmark. $\Delta P(a_j, f_i, k)_{pret}$ is generated at per core level because different cores might have different $\Delta IPC(t_i, f_i, k)$. Fortunately, the modern multiple processor supports per core level PMCs. According to the modern processor architecture, however, the formula needs to be modified because $P(f_i)$ accounts for the power consumed by shared resources that should not be replicated. One example of the shared resources is L2 cache. To recalculate it, we introduce another parameter that should be determined at the training stage, $P_{shared}(k)$. In order to retrieve information on $P_{shared}(k)$, we re-execute training benchmarks on k cores, and select median value as $P_{shared}(k)$ for each k. The values of $P_{shared}(k)$ are different, which is determined by the total number of cores utilized by a task simultaneously. The bigger k is, the larger $P_{shared}(k)$ could be. The final formula to estimate the power dissipation of a_j of a multicore processor is the following:

$$
\begin{aligned}
P(a_j, f_i)_{pret\_total} &= \sum_{k=1}^{k=cores} (\Delta P(a_j, f_i, k)_{pret} + P(f_i)) \\
&= \sum_{k=1}^{k=cores} (P_{inct}(f_i) + P_{\Delta}(f_i) * \Delta IPC(a_j, f_i, k)) \\
&+ \sum_{k=1}^{k=cores} P(f_i) - P_{shared}(k)
\end{aligned} \tag{4.6}
$$

Design Microbenchmarks

The power model we proposed decides which benchmarks we need. This is an important step because inappropriate choices will lead to inaccuracy. First, a wide range of IPC value needs to be covered by training benchmarks. It is extremely important to test two margins of benchmarks with smaller or larger IPC values since we observe different power behaviors affected by IPC at those ranges. Second, an even distribution of benchmarks according to their IPC values is preferred. In the power model, we divide training benchmarks into three groups based on IPC values. It is more informative if the number of training benchmarks resides in each group equally.

However, it is unrealistic to consider all cases especially we only use one PMCs. Even worse, there is no information about which subunit is exercising by only profiling IPC. For example, two workloads stressing integer and cache respectively probably have the same IPC values, yet the integer benchmark might consume less power than the cache operation does. Besides, the power dissipation is also affected by the inputs. The same FFT algorithm might produce more power dissipation for a larger input size. In conclusion, it is critical to generate proper training workloads covering a sufficient variety CPU activities for a linear regression based approach.

In our study, we implement totally 36 benchmarks exercising various CPU components, such as INT, FP, and BPU. In order to emphasize the simplicity and applicability of the power

Microbenchmark	Description	Approx. IPC
INT(1)	Arithmetic Integer Operation	0.50
INT(2)	Arithmetic Integer Operation	1.62
INT(3)	Arithmetic Integer Operation	2.65
FP(1)	Arithmetic Floating Point Operation	0.48
FP(2)	Arithmetic Floating Point Operation	1.12
FP(3)	Arithmetic Floating Point	1.43
Cache(1)	Cache line Reading	0.12
Cache(2)	Cache line Reading	2.15
BP	Brunch Prediction	1.00
IS	Insertion Sort Integers	1.77
ISF	Insertion Sort Floating Point	2.26
QUICK	Quick Sort Integer	1.01

Table 4.1: Training benchmarks suite.

model, we select 12 workloads covering maximum subunits, occupying a wide range of IPC values, and fairly even distributed. We list the benchmarks utilized in our study as Table 4.1. In general, the workloads exercise most of the processor subunits separately. The last three benchmarks utilize several components together to form mix benchmarks.

In the next section, we will discuss the method of relating the power behavior to the source code level profiling. Basically, based on the power model proposed, we design APIs locating source code function blocks according to the estimated power dissipation.

4.3 SPAN Design and Implementation

We are now in a position to automate the process of power profiling and correlate power dissipation to source code functions. We argue that it is crucial to design a PMC-based power estimator to association with source code based on two reasons. On one hand, it is convenient for software developers to identify their source code with actual power dissipation phases before any power/energy optimization. This will give developers more detail information of where their

power/energy optimization should target on. On the other hand, PMC-based approach is relatively easy to apply in reality. On the contrary, hardware instrumentation definitely offers high accuracy; however, in practical, this method is limited due to hardware requirements.

We design a tool, SPAN, to provide live, real time power phases information of running applications. Generally, given a power model, there are two methods to enable synchronization between power dissipation and source code. The first approach is run-time instrumentation at binary-level. This method usually has a high granularity control over the execution. Because our approach mainly assists developers, we adopt the second option that specifies a suite of external API calls to correlate power estimation with application source codes. We refer to it as source code level instrumentation. The advantages of this method include the following items: lower overhead, applicability, and independence against instrumentation tools, such as PIN [1]. However, our approach requires developers to add some code manually to call the SPAN APIs.

The basic flow of the SPAN tool is illustrated in Figure 4.4(a). The two inputs of SPAN are the application information and PMC values. At the application level, the app information and the estimation control APIs are passed to the control thread through the designed SPAN APIs. Utilizing the run-time PMC values by calling the system call, the analyzer thread applies the power model proposed in Section 4.2.2 to estimate the power dissipation. Finally, the SPAN outputs a figure of estimated power dissipation represented by different colors, such as Figure 4.4(b) shows.

In order to support the proposed mechanism, it is critical to provide a set of flexible APIs to applications. We show some of the designed SPAN APIs in Table 4.2. Currently, we implemented a preliminary C library of these APIs.

Given these APIs, the SPAN works as follows. First, we prepare a default file describing a set of power model parameters and an estimation frequency by calling span_c-
reate(). Once the targeting application runs, PMCs are opened for each core respectively by calling span_open(); then, a SPAN control thread, which stores the row PMC information and the application function information (e.g., function name and start time), is invoked before each profiling function. The recording continues until we call the span_stop or span_pause(). The

APIs	Description
`span_create()`	Prepare a power model profile which records basic parameters
`span_open()`	Initialize a SPAN control thread and targeting PMCs
`span_start(char* func, char* log)`	Record the targeting application function and specify the log file name
`span_stop(char* func, char* log)`	Stop the power estimation for a specified app function
`span_pause()`	Temporally stop reading PMCs
`span_continue()`	Resume reading PMCs
`span_change_rate(int freq)`	Shift the estimation rate, basically this methods control the PMC sampling rate
`span_change_model(float* model, File* model)`	Modify the model parameters in the model file according to the platform
`span_close()`	Close the opened PMCs and SPAN control thread
`span_output(char* log, FILE* power)`	Invoke SPAN analyzer thread and produce the detailed power estimation information with respective to the profiled functions to the destination file

Table 4.2: SPAN APIs.

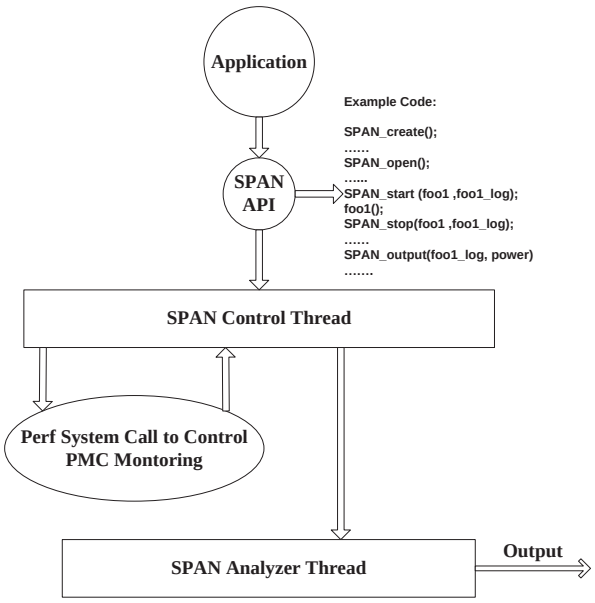
output is generated and stored into another file finally.

4.4 Validation and Evaluation

We mainly evaluate our approach in two categories. First we need to discuss the accuracy of the power model. The second part covers the evaluation of SPAN on the source code level power estimation.

4.4.1 Environments

Specifically, we evaluate the power model on two different platforms, $Asus\_intel\_4$ and HP_amd_6, where 4 and 6 represent the number of cores on each CPU respectively. The hardware configuration of each platform can be found in Table 4.3. We estimate the power generated by the SPEC2008Cjvm [130] benchmarks to validate the power model. We use Java version 1.6.0_18 on both platforms to launch each benchmark. The warm time is set to 5 minutes, and the iteration time is 10 minutes. We change $-bt$ option to change the number of threads. We plan to restrict the CPU affinity to one core during the training process originally, which will minimize CPU migrations and provide a set of more optimized model parameters, but the assumption of no CPU migration conflict with the reality. Therefore, the system does not restrict CPU affinity in all of our training and evaluation process. The PMCs values are collected using the kernel system call [32], _NR_perf_event_open(), which starts to be available



(a) The flow chart of SPAN.

(b) The example output of SPAN

Figure 4.4: Desgin of SPAN.

Platform	$Asus\_intel\_4$	HP_amd_6
Model	Asus Essentio CM5570	HP Pavilion Elite HPE-000
CPU	Intel Q8200	AMD Phenom
Core Frequencies	2.34GHz,2.00GHz	2.6GHz, 2.0GHz, 1.4GHz
# of cores	4	6
Memory	DDR3 6GB	DDR3 8GB
OS	Linux 2.6.31	Linux 2.6.31

Table 4.3: System configurations.

in Linux kernel version 2.6.31.

Leakage power becomes a non-trivial portion of the power budget on modern superscalar processors. Experimental results show that leakage current increases exponentially with the supply voltage [124]; however, given a specific CPU frequency and supply voltage, as the input of our model, the leakage power is fixed. Besides, our power model mainly focuses on the dynamic power dissipation generated by a given workload. Therefore, we do not incorporate the leakage power in our power model.

In order to minimize the temperature effect on power, after each valid run, we set 10 minutes as cooling time. The static power is measured before each execution, and we guarantee the variation of the static power is less than 5% so that the results are comparable. It is worth noticing that there only exists neglectable static power variation for different operating frequencies [27]. Meanwhile, we use hardware measurement to collect power dissipation information on the processor as well. The results are compared with the estimated power dissipation in the next section.

4.4.2 Power Model Evaluation

The first step of using our power model is to generate a set of parameters from the training benchmarks. Some of the detailed parameters we derived from the training process are listed in Table 4.4. We can easily observe that the effects of IPC on power drop considerably at both margins: the IPC below 1.0 and beyond 2.0.

We evaluate our model in terms of accuracy. More and more research on power estimation

System Settings	$P(f_i)$	$IPC(f_i)$	$P_{inct}(f_i)$	$P_{\Delta}(f_i)$	$P_{shared}(k)$	$IPCrange$
Asus Essentio CM5570, single-core, 2.36GHz	15.74	0.49	-1.28E-15	1.79	0	$0 \sim 1.0$
	19.47	1.28	-0.60	4.41	0	$1.0 \sim 2.0$
	21.52	2.21	1.50E-15	1.49	0	beyond 2.0
Asus Essentio CM5570, two-core, 2.36GHz	15.74	0.49	-1.28E-15	1.79	10.44	$0 \sim 1.0$
	19.47	1.28	-0.60	4.41	10.44	$0 \sim 2.0$
	21.52	2.21	1.50E-15	1.49	10.44	beyond 2.0
HP Pavilion Elite HPE-000, single-core, 2.6GHz	25.55	0.45	0.18	0.87	0	$0 \sim 1.0$
	27.26	1.35	-0.10	1.58	0	$0 \sim 2.0$
	27.50	1.99	0.06	-0.18	0	beyond 2.0
HP Pavilion Elite HPE-000, two-core, 2.6GHz	25.55	0.45	0.18	0.87	18.84	$0 \sim 1.0$
	27.26	1.35	-0.10	1.58	18.84	$0 \sim 2.0$
	27.50	1.99	0.06	-0.18	18.84	beyond 2.0

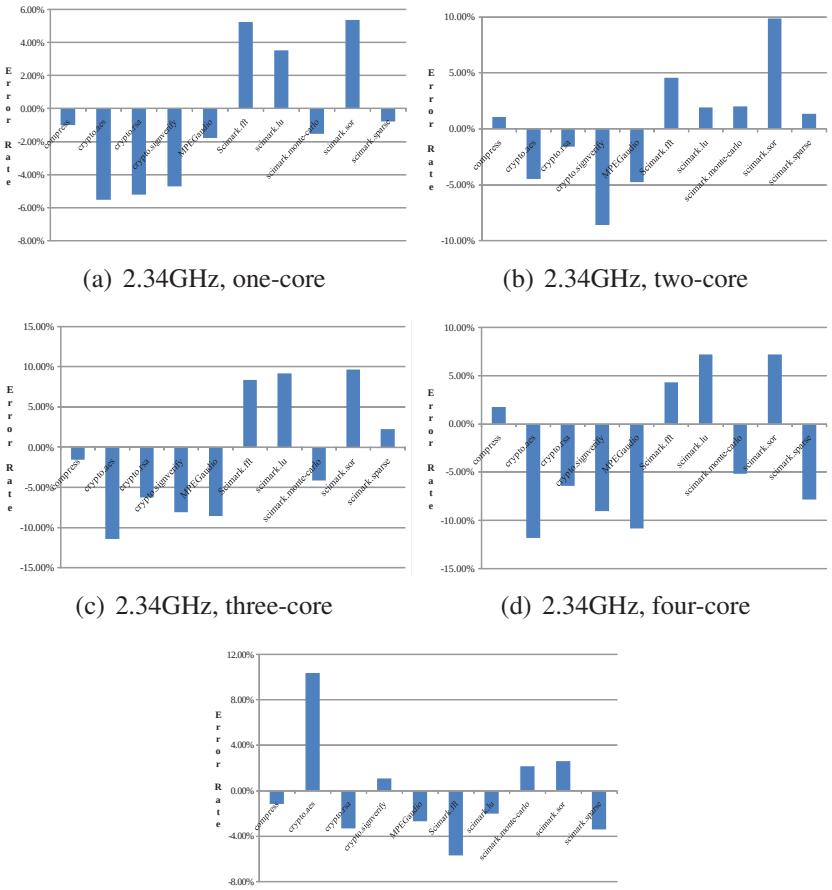
Table 4.4: Derived power model parameters.

techniques argues that accuracy is not the only aspect we should focus on [12, 49]. However, other characteristics, such as responsiveness, depends on acceptable accuracy. In addition, the power model usually provides reasonable responsiveness if it has high accuracy. We run SPEC 2008Cjvm benchmarks with multi-threads on possible frequencies to collect data. The errors are reported for the whole processor.

Through Figure 4.5(a) to 4.5(d) shows the percentage error from a single core to the maximum four cores running 10 different benchmarks on $Asus\_intel\_4$. As the figures illustrate, generally, there is an incremental relationship between error rate and the number of cores. The possible reason is that we do not consider the shared resource in a fine granularity in the power model due to the PMCs limit. In addition, the inter-core communications, which are another major source of power dissipation, cannot be captured by the power model simply deploying one PMC. Given such limited information, our model achieves 5.17% absolute error rate on average, with standard deviation of 5.40%.

Figure 4.5(e) summarizes the estimated error under frequency 2.00GHz on $Asus\_intel\_4$. Our model is able to achieve smaller error rate since the power dissipation for each benchmark decreases and falls into a narrow range, which is less unpredictable than the scenario of high frequency. The power dissipation of some particular benchmarks, such as $crypto.aes$, presents a low correlation coefficient to the IPC and extensive usage of other processor components, such as brunch prediction units.

Similarly, from Figure 4.6(a) to Figure 4.6(d), we report experimental results of our power



(a) 2.34GHz, one-core

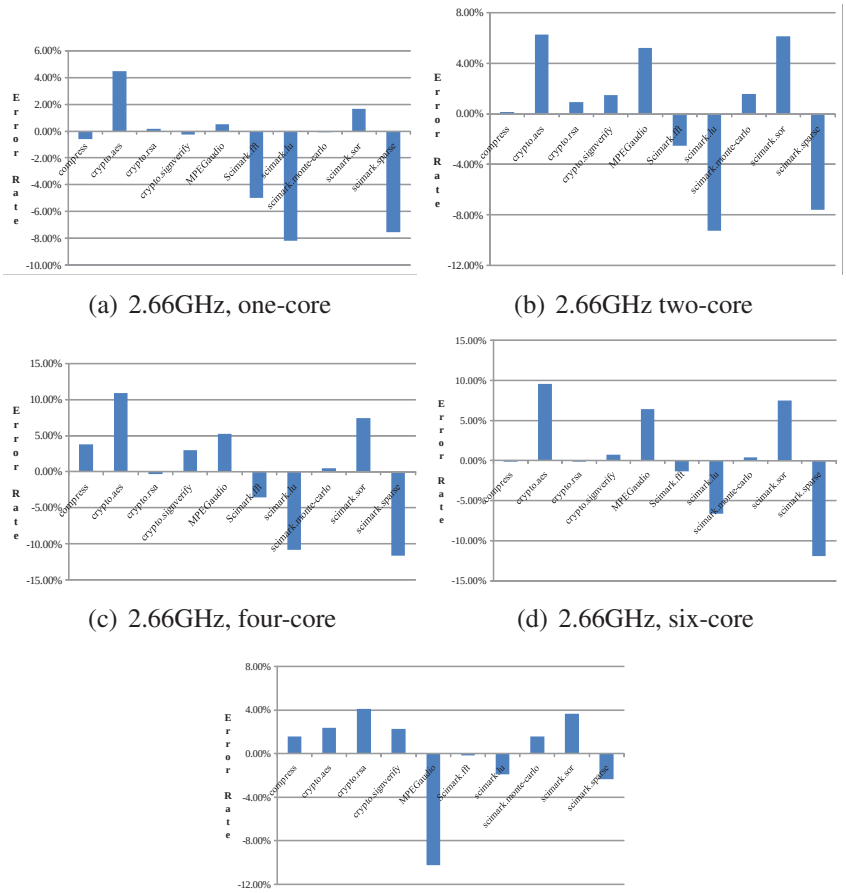
(b) 2.34GHz, two-core

(c) 2.34GHz, three-core

(d) 2.34GHz, four-core

(e) Summarized, 2.00GHz

Figure 4.5: Estimation error of SPEC 2008Cjvm on *Asus_intel_4*.



(a) 2.66GHz, one-core

(b) 2.66GHz two-core

(c) 2.66GHz, four-core

(d) 2.66GHz, six-core

(e) Summarized, 2.00GHz

model on *HP_amd_6*. We control the maximum and average absolute error rate to 11.26% and 4.46% respectively for up to the six-core scenario, with a vast majority of estimates exhibits very small errors. Besides, it is worth mentioning that our model does not consistently under- or over-estimate the power across the benchmark suite. We summarize the experiment results on frequencies of 2.00GHz and 1.40GHz in Figure 4.6(e) and 4.6(e) respectively, with average error rate of 3.14%.

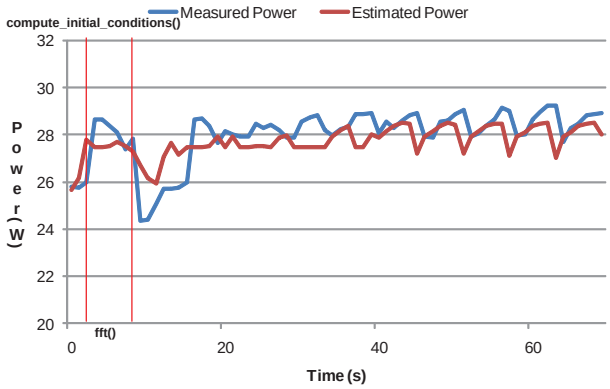
4.4.3 SPAN Evaluation

After illustrating the error rate of our model, in this subsection, we discuss the effectiveness of SPAN in details. As it was noted in Section 4.3, the SPAN is a source code instrumentation technique that keeps tracking power dissipation of each function block. We mainly evaluate

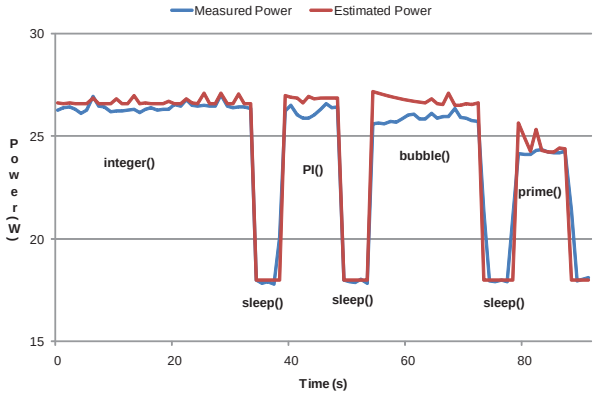
two aspects of the SPAN, the overhead and the responsiveness. We focus on two benchmarks for the testing purpose. One is the FT benchmark from NAS parallel benchmark suite. Another is a synthetic benchmark that we designed with the combination of integer operation, *PI* calculation, prime calculation, and bubble sort.

The overhead of instrumentation on both testing benchmarks is negligible. First, we measured the execution with and without the SPAN instrumentation for ten times each. The differences of execution time are within 1% on average. The reasons of low overhead are as follows: the instrumentation is at the source code function-level, which barely adds interruptions during executions; the PMCs used in the model are limited to the minimum values, which further reduce the computation and communication cost of SPAN. Second, we measured the power dissipation of the benchmarks with and without underneath SPAN threads that record counter values. The overall variance across the whole execution lies within 2% in ten valid runs. Considering other factors, such as temperature and power supply variation, 2% is a reasonable range in reality.

Though there is no standard method to evaluate the responsiveness of a power model, One of the simple and effective approaches are comparing the continuous measured and estimated power values. We utilize two multimeters storing the power dissipation of the target computer consistently into an assistant computer with the interval of one second. The benchmarks are executed on the $Asus\_intel\_4$ platform with the SPAN source code instrumentation to estimate the power. We plot the results in Figure 4.6. It is easy to observe that the estimated power is closely related to the measured power dissipation at the overall shape. We also mark the corresponding benchmark functions in each figure. The first iteration of benchmark FT mainly consists of two functions, $compute\_initial\_conditions()$ and $fft()$; then, the rest iterations follow the same procedure, which can be clearly observed from Figure 4.6(f). But the estimations present a certain level of delay due to the rapid function changes in the source code. Moreover, in Figure 4.6(g), we deliberately insert $sleep()$ function between each sub benchmark in the



(f) The FT benchmark with SPAN instrumentation.



(g) The synthetic benchmark with SPAN instrumentation.

Figure 4.6: Results of the SPAN evaluation on two benchmarks.

synthetic workload in order to distinguish each one of them easily. We achieve the error rate as low as 2.34% for the two benchmarks on average.

4.5 Related Work

Since we have already summarized a significant amount of work on power profiling, in this section we describe several previous efforts that are most related to *SPAN* from two aspects: *PMC-based power models* and *program power behavior analysis*.

4.5.1 PMC-based Power Models

Hardware performance counters are a set of special-purpose registers built into modern microprocessors to store the counts of hardware-related activities within computer systems.

Researchers often rely on those counters to conduct low-level performance analysis or tuning.

Frank Bellosa is one of the first proponents of applying PMCs to investigate the energy usage patterns and finding the correlation of hardware events and system power [10]. He uses information about active hardware units to establish a thread-specific energy accounting, and then he uses the power information for energy-aware scheduling policies. Kadayif *et al.* design a tool called Virtual Energy Counters (vEC), which is built on top of the Perfmon user library. Their power model mainly considers cache related performance counters. In [67], Isci *et al.* divide the processor into 22 function units and finds the relationship between the counters and those units. Although their results are very accurate, it is hard to be used on new platforms. G. Contreras and M. Martonosi [28] discover the power-IPC correlation and use five PMCs to estimate the power of workloads running on different CPU frequencies.

In [106], Powell *et al.* proposes a methodology to reduce the number of performance counters. They estimate the hardware activity events of several microarchitectural structures. Then, they associate the activity events with the power dissipation of such structures. Bertran *et al.* [12] demonstrate an alternative approach of using PMCs on CPU power estimation. Rather than directly deriving a power model using PMCs, they propose a method to treat each component of CPU separately, such as FE, INT and FP. Combining all the training parameters, they develop a fine-granularity power model. However, the training process is time-consuming to be extensively used in practical. In addition, the power model highly depends on the microarchitecture of the CPU.

Our main difference from all these works is that we combine the CPU frequency scaling and multicore features in the power model, which fits the trend of microprocessor design recently. Besides, our power model only employs one IPC. Other models [12] can achieve better accuracy and less variance compared with ours by collecting a number of counter values and training with more microbenchmarks, but barely can their models be applied to reality because of the model complexity.

4.5.2 Program Power Behavior Analysis

Understanding program behavior is at the foundation of computer architecture and program optimization [121]. As energy consumption becomes one of the most important design considerations, researchers also evaluate the power and performance during the software development period. Program power behavior analysis cannot only help us optimize the energy efficiency of the applications, but also help the systems intelligently schedule the tasks by using new power-aware scheduling algorithms [3, 77].

PowerScope [42] is one of the first work that map energy consumption to program structure. They develop a user-level daemon process and modify several system calls of the NetBSD kernel to sample process activity. Furthermore, they monitor energy consumption with collected data via a group of multimeters that connected to the power source. Finally, synchronization with the System Monitor is provided by connecting the multimeters external trigger input and output to pins on the parallel port of the profiling computer.

Similar with PowerScope, Ge *et al.* use their platform called PowerPack, a hardware-based power measurement and profiling platform, to analyze the application power behavior [46]. They insert a set of user-level APIs, such as pmeter_start_session and pmeter_end_session, before and after the code region of interest to map the power profile to the source code. Further more they analyze the power efficiency on multi-core platforms. The method they use to map power profile into program code is similar to our work; whereas, our approach is a pure software-based approach, and do not employ any hardware.

Isci *et al.* use the similarity matrix approach of [121] to deduce power phase behavior over the program runtime. Then they use component-based power breakdowns, computed by their power models, to identify power phases of programs. Their power model, however, is difficult to obtain because of PMCs limits.

4.6 Summary

Power measurements and profiling have already been studied extensively at different levels; however, more investigations are needed in the following two areas: improving power profiling techniques and using these strategies in power-aware software design.

Accuracy is not the only important requirement for power measurements and profiling. We envision that simplicity and adaptability are also very interesting aspects. Simple power models are needed to supply live power information for systems, otherwise the overhead will be too high to be used. In addition, as the number of cores on a single chip keeps increasing, on-chip network fabrics become one of the main power dissipation resources. Thus, future research needs to consider this unit and reevaluate the power indicators that are currently used. Furthermore, we still need to break down the power dissipated on shared resources such as caches, and find suitable indicators to break down higher level power information into lower levels.

In this chapter, we present a novel practical power modeling method based on performance monitoring counters (PMCs) by employing one PMC and 12 training benchmarks on two recent multicore processors. Based on the model, we design and implement SPAN to map the run-time power dissipation to application functions. We evaluate both the power model and SPAN on two modern multicore systems. Despite the limited information provided by only one PMC, we achieve an absolute error rate of 5.17% and 4.46% on the two platforms by using benchmarks from SPEC2008Cjvm suite. In addition, it shows fairly stable accuracy under different frequencies. We also collect empirical data to validate the SPAN tool. Using the FT benchmark from NAS Parallel benchmark suite and the synthetic workload, we reach accuracy as high as 97% on average. We achieved our second goal in this chapter.

Though the power model and SPAN proposed in this chapter is able to provide function level power dissipation information, especially when the target functions are available, SPAN has its limitation when facing the challenges during the instrumentation and profiling process.

For example, if the target functions are unknown, manually instrument every single function in a workload is not practical. In order to overcome those challenges, we introduce Safari, an automated profiling process in the next Chapter.

CHAPTER 5

SAFARI: FUNCTION-LEVEL POWER ANALYSIS

While in the previous chapter, we propose SPAN, which provide a set of APIs to generate function level power profiling results. However, SPAN has its limitations. In this chapter, we describe and implement an application function (subroutine call) level profiler, Safari. It can be used to generate power profiles of each function in an automatic manner. The experiment results using NPB parallel benchmark suite show that Safari is able to collect function level runtime information with overhead (16% on average) comparable to gprof. The power profiling results can be used for code optimization, power-aware scheduling, or even computing resource billing for future research.

5.1 Introduction

Different from hardware and system design and analysis, the impact of software on the power dissipation of a computer system has been overlooked. In fact, as the user of hardware resources, software has equivalent or even more effects on the power dissipation of a whole system. For example, Pathak *et al.* introduce a new type of bugs, energy bugs or ebug [101], on smartphones. Their results show that 35% of energy bugs stem from software, either the OS or the applications. Nevertheless, the authors pointed out narrowing down the root causes of ebugs to a software component is one of the crucial steps to fix energy bugs. A recent study shows that software bloats introduce excessive resource usage in large software systems [13] as well. Better understanding of software behavior associated with resource usage is crucial to detect similar scenarios. In addition, workload phases provide interesting information for performance optimization and they are usually related to functions/methods [47]. Function level power profiling is supposed to reveal power behavior along with resource usage information. These information would help developers to understand and leverage power dissipation in a

computer system from a new angle. However, information scarcity of dynamic power dissipation impedes developers' ability to produce more energy-efficient software. Even though the latest processors based on Sandy Bridge or Ivy Bridge Architecture provide power information from hardware counters, the power dissipation within each program block is still unclear. In addition, power model based approaches are still effective to estimate the power dissipation of other components, such as memories.

Regardless of the potential influence of software on power dissipation, its impact is usually underestimated. For example, most profiling tools are used to measure performance rather than power or energy. In addition, among the few available tools that estimate power dissipation, most of them do not consider the control flow of a program, which loses the insight of the execution of a program. Given such scarce information, it would be difficult for developers to evaluate or optimize the power usage of their programs.

Run-time profiling techniques usually deploy mechanisms to collect information from the target systems. Typically, in order to obtain more detailed results, a profiling process generates overhead. Consequently, a profiling process could disturb power measurement. In addition, the inaccuracy due to overhead can be enlarged because the collected data for power profiling usually need to be processed by power models. For example, Linear Regression is a commonly used technique to generate estimated power from collected run-time information [15]. Moreover, the sizes of applications are growing rapidly, which posts more challenges to analyze software power behavior.

In this chapter, we present a software function level power profiling tool, Safari. The goal of Safari is to provide function level power analysis while minimizing profiling overhead. In order to use Safari, first, we compile a target application to insert instrumentation code for each function. Then, run-time information is collected for the resource usage during the execution of functions. Finally, we apply an off-line analysis based on a selected power model to generate function power profiles.

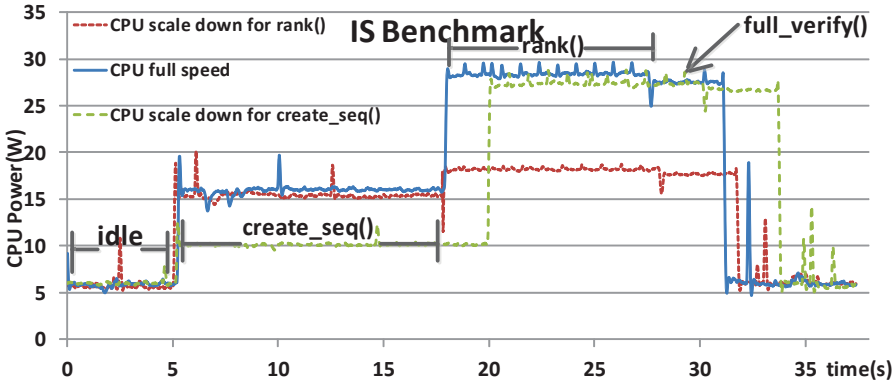


Figure 5.1: Power dissipation of IS.A on a Intel Core2 Quad 8200 processor.

The rest of the chapter is organized as follows. We start the chapter by presenting a motivating example in Section 5.2. In Section 5.3, the design considerations of Safari are described, followed by the evaluation results shown in Section 5.4. Related work is discussed in Section 5.5. At last, we describe future work and conclude the chapter in Section 5.6.

5.2 Motivating Examples

Usually different functions in a program have distinct power behavior. For example, we retrieve the function profile of IS.A benchmark from NPB 3.0 benchmark suite. There are three major steps in IS.A: `create_seq()`, `rank()`, and `full_verify()`. The power dissipation of IS.A is closely related to the three major functions as shown in Figure 5.1. We use two 0.005 Ohm current sense resistors (CSR) series connected to the 12V cable from a standard ATX2.0 power supply. The CPU current is measured by reading the voltage on the resistors using a NiDAQ 9205 unit and dividing the resistor value. We can calculate CPU power dissipation using the measured current and voltage values.

Given distinct power dissipation information along with application execution, one of the usage of power profile is to guide run-time power management. In this example, `rank()` function, which produces approximate 0.15 Instructions Per Cycle (IPC), is less CPU-bound during its execution (IPC values are used broadly as CPU power model input [137]). Systems could provide more fine-grained power management or scheduling schemes if resource usage

information can be retrieved beforehand. For example, we are able to use DVFS to scale down CPU frequency from 2.34GHz to 2.00GHz during the execution of `rank()`. As a result, we achieved 24% energy saving with 3% performance loss. On the contrary, 10% energy saving is achieved with 10% performance loss if we scale down CPU frequency during the execution of `create_seq()`, which has a higher average IPC value during its execution (around 0.6). In addition, we observe multiplication operations are intensively used in the source code of the `create_seq()` function, while the `rank()` function mainly contains branch-prediction and data movement operations. Hence, it is possible to utilize profiling results to guide system power management in a fine-grained fashion.

Based on this example, we observe that software characteristic is an indispensable part to analyze the power dissipation of a computer system. Safari attempts to accurately estimate the power dissipation of a function. Our rationale of using function level profiling includes the following aspects: first, application subroutines/functions are the basic units of executing tasks; second, function level profiling guarantees appropriate scale for optimization: coarser than the instruction level yet finer than the process or thread level; third, as the module design is one of the common methods to develop large scale software, function level power profiling fits this pattern well.

5.3 Method

Given the fact that power dissipation of function invocation is one of the major break points to understand software dynamic power dissipation, it is worth developing a profiling mechanism to generate function level power profile. The goal of Safari is *automatic power profiling based on per function resource usage with restricted overhead.*

In order to achieve this goal, there are three major points need to be considered. First, profiling overhead must be minimized. Although functions can be treated as the basic units to generate a profile, a majority of functions only accomplish tiny tasks, such as printing timestamps or reversing a string, which hardly present any potential for optimization or tuning in

most cases. However, profiling them consumes as much system resources as profiling major functions. For example, a `create_seq()` function invokes `randlc()` millions of times in the IS benchmark. As a result, instrumenting and profiling `randlc()` function produces much more overhead than profiling `create_seq()` along. Moreover, function power behavior varies according to different input data and execution paths. It is important for function level power profiling to reflect those characteristics. Additionally, the core part of power profiling is power models [12], which usually utilize system information and Performance Monitoring Counters (PMCs) as input. In this case, the collected information has to be associated to each function in an application.

5.3.1 Overview

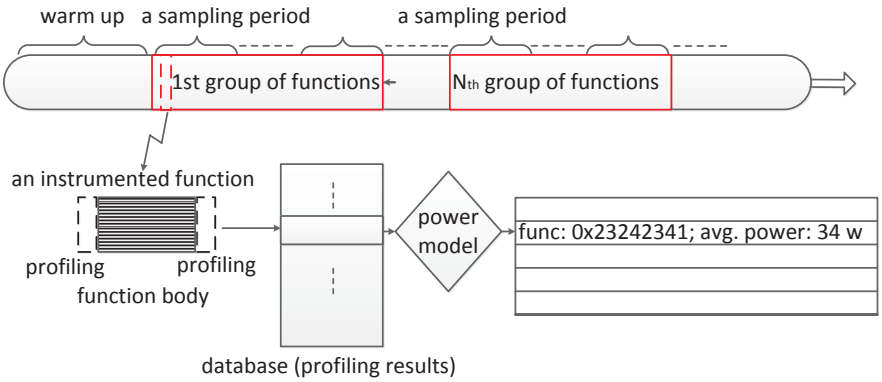


Figure 5.2: An overview of the profiling process.

The proposed profiling procedure is demonstrated in Figure 5.2. First of all, the execution of an application is divided into different parts. During the warm-up period, no profiling data are collected since usually only start-up activities, such as initializing some buffer, are executed during this period. The rest of execution is divided into different sampling periods. During a sampling period, only a certain number of selected functions are profiled. As a result, functions are randomly grouped into several categories. Only one instance of a function is profiled even the same function can be executed more than once during the same sampling period. Target function groups are switched as time elapsed. Safari collects data exactly before and after a

function being executed. Off-line analysis generates power profile based on a predefined power model. We will discuss the details of the profiling procedure in the following sections.

5.3.2 Function Level Power Profiling

Instrumentation Automation To instrument an application, there are two commonly used methods. One is to design a set of APIs that control the procedure of data collection at run-time [137, 46]. The other approach is automatic instrumentation using some available compilation tools, such as PIN [1]. In a considerable large program, there can be more than ten thousands function prototypes. The goal of function level power profiling is to locate the relative power hungry part of source code. Each function block can be a candidate if we treat the whole program as a black box, which means all of them need to be considered. It takes excessive human efforts if we instrument each function manually. As a result, we adopt the second approach because of its simplicity to developers. Whereas, automatic instrumentation has its disadvantage such as it dose not distinguish major functions and trivial ones. If we simply apply this technique, the profiling process could cause unnecessary overhead. Safari adopts several techniques to overcome this effect.

In the implementation of Safari, we use a function instrumentation utility designed for GCC compiler, `-finstrument-functions`, to insert two profiling functions that will be invoked at every entry and exit of each function, namely, `__cyg_profile_func_enter()` and `__cyg_profile_func_exit()` as illustrated in Figure 5.2. At run-time, in addition to execute instructions in a normal function body, two profiling functions are attached to the both ends of a function to collect *per thread resource usage information*.

Warm up Usually a system is not stable during the warm-up phase of an application. For instance, buffers need to be initialized. In order to get accurate profiling results, the data collection for profiling starts after a warm up period as shown in Figure 5.2. The total length of warm up depends on a specific program and is tunable at start-up.

Overhead Reduction As described in the previous paragraphs, in order to use automatic

instrumentation properly, the key issue is reducing instrumentation overhead. A power model cannot be accurate if overhead dominate the collected data. There are two major concerns associated with function level power profiling. First, a specific function can be invoked many times during a relative short duration. This scenario affects not only power profiling of the function itself but also other threads. For instance, context switch or other system activities might rise. Second, nested function calls will add more inaccuracy to outsider ones. Automatic instrumentation in Safari is based on insertion of two additional function calls at the entry and exit of each function. If a function has many nested function calls, the collected model input can be misleading (dominated by the inserted profiling functions).

The solution to the first problem is limiting the instrumentation of same functions. If a function is invoked many times, Safari only samples one instance in order to eliminate the overhead of repeated profile. However, this method has a potential problem: if the code path in this specific function is changed due to different parameters or input, the power dissipation of this function will also change. We solve this problem by using multiple records. In the implementation, we use *bloom filter* [16] to record functions that have been profiled. Before a function is to be profiled, Safari checks the *bloom filter* first. This method is named Safari_1 in the rest of the chapter. We demonstrate this idea in Figure 5.2. During a sampling period, the same function call is only profiled once by checking the bloom filter. Although *bloom filter* is able to control which function to be profiled, checking bloom filter itself consumes system resources. Normally this part of overhead is acceptable unless an application has an extremely high rate of function calls. In this case, checking bloom filter could dominate the profiling process.

The total number of instrumented functions needs to be controlled in order to solve the second problem. The overhead produced by nested function calls can be reduced if only a set of selected functions are profiled during a certain period, which means other functions execute normally without profiling. In addition, profiling module by module for a large program (for

example, an application might contain over 10,000 functions) is extremely useful in order to generate accurate power profiling results. Figure 5.2 shows that only a group of functions are profiled for several sampling periods. By adjusting group sizes, overhead can be effectively reduced. However, some functions might not get profiled if this method is used. Commercial software, such as base station controller, is deployed to run for a considerable long period (months or years). Statistically, most of functions can be profiled in such a setting. This strategy is refereed to as Safari_2 in the evaluation. Functions can be grouped alphabetically or according to their addresses. By adjusting the group size and the total length of the warm up period, Safari is able to generate power profile for most of functions.

Multiple Records For a frequently invoked function, we should profile it multiple times in order to generate correct power profiles because the code path of an application might vary. As aforementioned, a function is only sampled once in a sampling period. The result can be inaccurate if the code path changes afterward. The solution is profiling the same function during different sampling periods as Figure 5.2 shows. Statistically, random sampling represents characteristics of the whole sample space if the function has been invoked multiple times and the execution is sufficient long (high confidence level). For example, random sampling can be used to approximate the percentage of a path occurrence of a function with *"if" or "switch" statement* in it if the profiling process is sufficient long.

Power Model In order to collect the input data for a power model, the following aspects need to be considered. 1) availability: the input data should be easy to collect. Sometimes system features constrain the data that are able to be sampled for a system. For example, usually two PMCs values can be collected simultaneously given the limited number of hardware registers [132]. 2) complementary: the collected input data are most useful when they cover a certain range of system events. Given the fact that the total amount of data can be collected is limited in order to reduce overhead, it is important to explore the resource usage of a system as much as possible. For instance, cache miss rate and bus transaction rate contains some over-

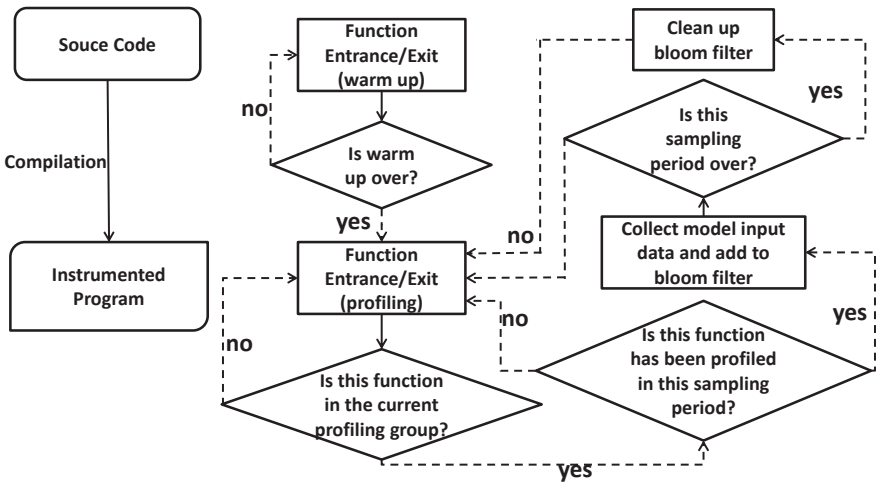


Figure 5.3: Run-time profiling and information collecting.

lapping information because, most likely, CPU retrieve data from memory through bus if the data cannot be found in the last level cache. In this chapter, we use OS level metrics and PMCs as the model input:

- CPU utilization: it represents the average CPU usage during the execution of a function. The value can be retrieved from Linux PROC File System.

- Last Level Cache miss rate: it partially quantifies how frequently memory has been used for read and write. The value can be retrieved from PMCs.

- Context switch rate: we use the context switch rate to estimate the overhead of running multiple processes in a system and attribute it to the functions in threads, during which context switches occurred. The value can be retrieved from Linux procfs.

- Instruction per cycle (IPC): we use IPC values to calculate the effectiveness of a CPU. A strong relationship between IPC and power dissipation has been revealed in several articles [84, 137].

- CPU frequency: the frequency of CPU is directly linear related to power dissipation. The value can be retrieved from cpufreq subsystem from Linux.

In order to construct a power model, usually a mathematical method such as linear regression or nonlinear regression is used. Given the model input, we use the following equation to estimate power dissipation: $P = a_1 \times cpu_u + a_2 \times cache + a_3 \times cs + a_4 \times ipc + a5 \times cpu_f + P_{idle}$, where $\{a_1 \ldots a_n\}$ are coefficients to be determined by a set of training benchmarks. The power model is not the major concern of this chapter. The model can be substituted with other models.

The method is summarized in Figure 5.3. Given the source code, 1) we compile it to generate an instrumented version of executable. The source code needs to be compiled with `-finstrument-functions` option. 2) The compiled object files are linked to a static library provided by Safari, *libsafari.lib*. 3) Instrumented program collects run-time function resource usage information. There is no information collected during the warm up period. Then, the instrumented program determines if the encountered function has been profiled or not by checking the bloom filter during one sampling period. In addition, functions are divided into groups to reduce profiling overhead as well. The collected data are the input of the power model.

5.4 Evaluation

We mainly evaluate the effectiveness of Safari and the overhead introduced by function level power profiling. The experiment platform contains a Intel Core 2 Quad 8200 CPU with 6GB memory. The processor is able to work on two frequencies, 2.00GHz and 2.34GHz. All the results in this section are generated by setting CPU frequency to 2.34GHz. We use a NiDAQ 9205 unit to record the CPU power dissipation from the 4 pin power supply on the motherboard. The original sampling rate is 1KHz. IIR low bandpass filter is utilized to filter noise. Data are re-sampled at the rate of 50Hz. We mainly use NPB3.0 benchmark OMP implementation as the target applications.

First, we use Safari to sample CPU activities and produce CPU power profiles. We utilize linear regression to construct the power model based on training benchmarks. The power model is not the major concern because Safari is flexible to use different models and run-time information. In order to obtain a stable external power measurement, we deliberately execute

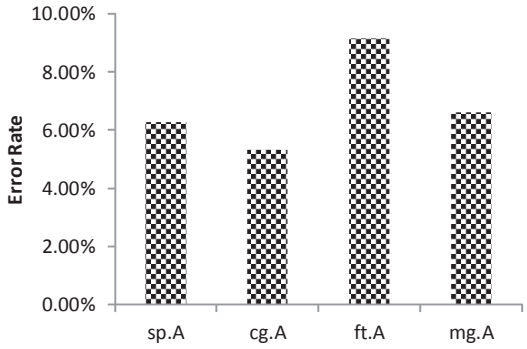


Figure 5.4: Estimation error rate of CPU power for different functions, the function names are shown in Table 5.1.

Table 5.1: Activities inside the functions.

Benchmark	Function	IPC	CPU utilization	Cache miss rate
SP.A	compute_rhs__	1.33	97	1162.15
CG.A	conj_grad__	1.28	98	1603.67
FT.A	fftxyz_	2.08	98	438.06
MG.A	mg3p_	1.45	99	1045.28

the target functions into a infinite loop. The error rates are demonstrated in Figure 5.4. The CPU power estimation has an average error rate of 6.85% for the selected four benchmarks. The detailed activities inside each function are shown in Table 5.1. The accuracy of the collected activities is closely related to the system resolution. For instance, CPU utilization is obtained from PROC File Systems, which usually utilize jiffy as the basic unit. No correct information could be retrieved if a function's execution time is beyond that resolution. However, this constraint does not affect most major functions.

Table 5.2: Profiling overhead with Safari_1.

Type	Benchmark	Overhead (8/1)	Overhead (4/1)	Overhead (2/1)	Overhead (1)	Overhead (gprof)	Call rate (calls/sec)
SER	*CG*	9.89%	6.91%	6.18%	6.04%	1.03%	68989.07
	MG	1.89%	0.52%	0.78%	0.31%	0.21%	99.26
	FT	1.49%	1.00%	0.93%	0.65%	0.50%	2719.37
	EP	0.45%	0.27%	0.05%	0.33%	0.47%	0.33
	LU	1.08%	1.01%	0.98%	0.75%	0.35%	14591.35
	SP	11.09%	10.89%	10.85%	10.72%	1.29%	102004.79
	BT	288.94%	288.14%	287.27%	286.48%	5.36%	3706727.08
	IS	908.28%	907.25%	905.39%	911.84%	15.23%	11158773.53
OMP 4 threads	*CG*	42.21%	21.04%	11.90%	18.28%	47.54%	38134
	MG	72.36%	39.02%	24.64%	15.54%	2.67%	175.93
	FT	41.44%	32.04%	33.77%	20.99%	21.24%	77491.14
	EP	1.34%	1.32%	1.21%	0.94%	0.39%	0.17
	LU	2.11%	2.10%	1.78%	1.55%	1.55%	6347.21
	SP	3.66%	3.48%	3.33%	3.03%	1.93%	15996.28
	BT	309.58%	298.44%	292.87%	284.51%	63.56%	4936017.01
	IS	508.08%	499.23%	490.79%	495.19%	124.48%	2905743.91

Next, we measure the overhead introduced by Safari. Because Safari has two policies to

reduce profiling overhead, (one sample for a function inside a sampling period and different function groups), we first only deploy the first mechanism, Safari_1. Functions are not divided in Safari_1. The profiling results are shown in Table 5.2. The number of sampling periods is also a factor affecting overhead since more samples for each function can extend total execution time. Therefore, different sampling periods are used in this evaluation. For example, the column labeled with *overhead (1/8)* means that there are 8 sampling periods totally during the execution. In other words, maximum 8 samples can be collected for each function during application execution. The overhead is measured as execution time when we explore profiling techniques.

As Table 5.2 shows, the overhead generated by Safari is comparable with that of generated by gprof in most cases. As expected, the overhead increases slightly as the number of sample collected increases. Overall, the overhead generated by OMP version of benchmarks is higher compared with SER cases for both Safari and gprof because the contention of recording information in one single file for multiple threads. It is obvious that the overhead generated by BT and IS benchmark is as high as 546% on average for SER and OMP benchmarks. The root reason is because these two benchmarks have extremely high function calls rates. On average, the function call rates of BT and IS are 203 times higher than that of the rest five benchmarks. The BT benchmark has nested function calls that generate excessive overhead.

We deploy both Safari_1 and Safari_2 to reduce overhead for BT and IS benchmarks, especially. The number of sampling periods is denoted as n. We divide functions in a workload into m groups, where $m \in [1,n] \wedge n = am, a \in \mathbb{Z}$. If $m = 1$, the effect of Safari_2 disappears. This setting is for simplicity. The values of m and n are more flexible if execution time is long enough. We evaluate Safari_2 on BT and IS benchmarks with at most one sample is collected for each function. The results are shown in Table 5.3. For a fixed m value, as the n increase, the total overhead increases as well since more samples are collected. If n is fixed, the total overhead drops as m doubled because trivial functions might not be profiled with a bigger m

value. However, major functions in both benchmarks are profiled because they usually iterate for more than one time.

Table 5.3: Profiling overhead with Safari_2.

Type	Benchmark	n/m	2	4
SER	*BT*	4	29.62%	26.45%
		8	56.81%	42.32%
	IS	4	16.86%	18.62%
		8	24.30%	23.34%
OMP	*BT*	4	57.49%	45.65%
		8	82.42%	54.21%
	IS	4	14.14%	12.45%
		8	28.4%	20.60%

In order to further measure the profiling overhead, we let Safari to profile only one function repeatedly. The results are used to compare with the execution time without profiling. Besides the aforementioned platform, we use a Cavium 6300 evaluation board as an example of embedded systems. The board is equipped with six cnMIPS II processor cores, 4GB DDR3 memory and some other co-processing units such as compressor and encrypter. The results are shown in Table 5.4. The profiling overhead means that functions are instrumented and model input data is collected. To profile a function once introduces about 0.8ms overhead on Cavium 6300 evaluation board. While, if a function is only instrumented without actual profiling (for example, the program encounters a function that has already been profiled during a sampling period) consumes much less overhead. Both of them is neglectable compared to a function body conducting 512*512 matrix calculation which takes few seconds. In addition, as we avoid frequent profiling in a given sampling period, the overall overhead is under restrict control.

Table 5.4: Profiling overhead

Platform	Profiling overhead (sec)	Instrumentation overhead (sec)	512*512 matrix mcl (sec)
Cavium 6300	$8 * 10^{-4}$	$5 * 10^{-7}$	7.4
Intel Core 2	$4 * 10^{-4}$	$2 * 10^{-7}$	1.4

5.5 Related Work

Software power dissipation is directly related to dynamic power, which becomes an increasing portion under the context of energy-proportional computing. However, only few research projects focus on software power analysis.

Although system level power management has effectively been investigated in recently years, there is a realization that software has dramatic impact on power dissipation. Therefore, in-depth understanding of software power dissipation becomes one of the major consideration while designing power-aware systems. Ge *et al.*, propose PowerPack [46] to generate component level power profiles. This approach targets on the cluster level. PowerPack provides APIs to synchronize external power measurement and function execution of the target application. However, manual instrumentation is inconvenient for large scale applications. Hänig *et. al*, propose SEEP [58], which uses symbolic execution to explore possible code paths and entities in a program and to generate energy profiles for a specific target platform. Instruction level energy profile is needed for each platform in advance in order to generate energy profiles for a program.

Moreover, as the energy consumption and power dissipation of a computer system stem from the interplay of hardware and software, they must be considered equally important. Bhattacharya *et al.* propose an analytical model to estimate energy cost of software bloat on a specific platform [13]. The results show that reducing software bloat can achieve as much as 40% energy saving. However, the study shows both hardware and software need to be considered to improve energy efficiency.

5.6 Summary

In this chapter, we achieved our third objective. Basically, based on the previous chapter, we propose a function level power profiling tool, Safari. It can be used to associate run-time resources usage with the execution of application functions. The experiment results show that Safari is able to produce function level profiling with limited overhead (on average 16%

overhead if maximum one sample is collected for each function). It can be used to connect application activities to hardware for energy-efficient design, such as application aware power management and fine-grained scheduling.

So far, we mainly discussed about power profiling techniques with models and tools developed. All efforts are for one goal: to reduce the energy consumption. Starting from next chapter, we recheck the energy-efficiency for a system during the execution in a system. Specifically, using the proposed analyzing tools, we examine the factors that impact the system energy efficiency during the execution of a workload.

CHAPTER 6

CPT MODEL

In the previous chapters, we present the analysis and tools to analyze system dynamic power dissipation triggered by active workload. In order to use the exposed workload information to help system achieve energy efficiency, in this chapter, we take the initial step and define a general energy-efficiency model, the CPT model, for multi-core computer systems.

Before any optimization is in process, we observed that it is necessary to obtain a general metric that represents the energy efficiency of a computer system, for a specific configuration, given a certain amount of workload. CPT is a unified model that helps to decide the near-optimal configuration of a system in terms of energy efficiency to execute a given workload. In addition, we expect the model can be utilized to analyze possible knobs that are used to improve energy efficiency. Three case studies are employed to illustrate the usage of the proposed CPT model.

6.1 Introduction

The conventional computing area is dominated by the pursuit of performance. The community has not realized the importance of energy efficiency until recently [18]. As a result, energy-efficient techniques have been used across different layers in almost every single system, ranging from a single chip to a large data center. These techniques include low-power circuit designs, tunable device states (Dynamic Voltage and Frequency Scaling), dynamic power management from operating systems, and energy-efficient software. Although current computer systems already achieve much higher efficiency ratings compared with that of previous generations, there is still potential headroom for improvement.

With the development in energy-efficient computing, one of the fundamental questions is how to define a model to represent energy efficiency. An appropriate energy-efficiency metric

not only can be used to evaluate and analyze existing techniques, but also helps to explore new techniques in this field. However, defining a energy-efficiency model is challenging. For one thing, the model should be sufficiently general in describing various techniques. Commonly used knobs such as multi-threading and DVFS have to be meaningful according to the model. Optimizations from different layers have to be expressed at some level from the model as well. For instance, the model is better to convey the idea of both clocking gating [142] and workload consolidation [133].

Moreover, energy efficiency has to be associated with workload characteristics. For example, requests per second or transactions per day is the metric that typically is used to measure the throughput of web service applications, while million instructions per second (MIPS) is one of the most interested performance indicators in scientific computing field.

In this chapter, we propose a general energy-efficiency model, CPT, which enables efficiency analysis of a system, given a running workload. The rest of the chapter is organized as follows: we start the chapter by presenting the CPT model and analyze each component in the model in Section 6.2, followed by case studies in Section 6.3. Related work is discussed in Section 6.4. Finally, we summarize the chapter in Section 6.5.

6.2 The CPT Model

In the CPT model, given a fixed amount of workload, we ask the question how much energy is consumed in order to complete the task. Specifically, the model is represented as follows

$$\begin{aligned} \mathscr{E} = Workload/Energy &= \frac{W}{(P_{AI} + C \times P_t)T} \\ &= \frac{W}{P_{AI} \times T + C \times P_t \times T} \end{aligned} \tag{6.1}$$

where $\mathscr{E}$ stands for energy efficiency. W represents the total workload size that is assigned to a system. P_{AI} and P_t denote active idle power of the system and average power dissipation of each thread, respectively. Specifically, P_{AI} is the power dissipation while a system is idle in $C0$

with the corresponding $P0$ state specified in ACPI standard [2]. C indicates the concurrency level of the workload. Intuitively, the more concurrency threads that are used, the quicker a job can be completed. System power dissipation, however, rises with more system resources being used. The last factor, T, is the total time taken to complete the workload. The name of CPT was conceived using the three most important parameters, concurrency, power and executive time.

In order to improve the overall $\mathscr{E}$, each part in Equation 6.1 should be considered. In reality, changing each item usually subsequently alters other factors in Equation 6.1. For example, improving performance reduces T; however, in order to improve performance, usually, active power increases. To be clear, we argue that it is more important to compare the energy efficiency of the same workload using different designs and/or implementations.

6.2.1 Workload (W)

Given the other factors fixed, in order to improve $\mathscr{E}$, intuitively, we can assign more workload to a system as much as possible. These scenarios can be found in data centers, where facility power dissipation is limited. At this level, the concurrency can be roughly estimated as how many nodes have been deployed in a data center. Hence, it is better to operate a facility to its upper capacity limit so that energy efficiency can be maximally guaranteed.

Fan *et al.* propose several ideas to improve $\mathscr{E}$ in [40]. Through in-depth analysis, the authors discuss possible capacity that can be safely incorporated into different layers, which include racks, PDUs, and clusters. Although in this process, there are more nodes added into the system, P_t can be reduced via DVFS and idle state so that the denominator in Equation 6.1 remains within the total power limit of the facility. The basic idea of techniques in this category is to accomplish more jobs while keeping the product of C and P_t unchanged.

6.2.2 Concurrency (C)

As multi-core platforms become common on servers and even smart phones, implementing concurrency applications generally will help to improve performance. By assigning each piece

of a job to different cores, system resources can be efficiently utilized. Most likely, if a job can be finished earlier, its $\mathscr{E}$ can be improved as well because T is reduced. However, it is not always the case. The well-known concurrency hazards are a collection of problems that could possibly occur if concurrency is not implemented properly. The hazards include false sharing, memory contention, incorrect granularity, and so on. On the one hand, the execution time, T, could be increasing. On the other hand, the total P_t might rise which in turn sabotages $\mathscr{E}$. False sharing is a typical problem that can be found in multi-threading applications with shared memory. In the case of false sharing, a certain amount of cache lines are being swapped in and out from a cache frequently, which invokes additional power dissipation and extends the execution time. In this case, P_t and T are both affected.

Speedup models are supposed to estimate the benefits in terms of execution time by using more threads to work on a job [46]. Normally, allocating more cores to a job has dramatic benefit on execution time if it is used properly. For example, embarrassingly parallel applications benefit the most from multi-core architecture theoretically because there is no dependency between paralleled tasks. A typical example is that a GPU has a much larger number of cores (from 500 - 900) compared to that of a general purpose CPU. However, most other parallel applications do not hold the assumption that there is no dependency between tasks. Communication between tasks becomes the primary concern when the number of cores increases. Consequently, the bottleneck of finishing a job shifts from computation needs to communication demands. The speedup effect diminishes while the concurrency level still ascends, eventually decreasing $\mathscr{E}$ [46].

The concurrency level, C, influences the overall $\mathscr{E}$ in various ways. Its effect is a complex combination of system architecture and workload characteristics. Optimal concurrency level from a performance perspective does not necessarily indicate the maximum $\mathscr{E}$. Selecting an appropriate C becomes a more complicated problem in power-aware computing setting (DVFS-enabled).

6.2.3 Active idle power (P_{AI})

The active idle state is the normal operating state of a system when it is idle, or according to ACPI standard, the $C0$ idle state. No wake up is required before a system executes jobs. In active idle state, power has been utilized to maintain the operation state of a system.

Most techniques used to improve $\mathscr{E}$ by reducing P_{AI} are at circuit level. The idea is to reduce leakage power. Usually low power design devices can be used to achieve this goal. As the density of transistors on a die increases tremendously, the static power dissipation of a processor occupies a large portion of the total power. The reason is because leakage power rises as more transistors are put on a chip. Low power devices usually sacrifice some performance to achieve less power dissipation.

One well-known strategy, "race to idle" [128], states that a system should finish its job as quickly as possible and rush to an idle state. This is partially because $\mathscr{E}$ can be improved by reducing T. In addition, a system could enter deeper C states.

One of the most beneficial results that comes from reducing active idle power is that it almost has no effects to other components in Equation 6.1. Therefore, it can be safely used together with other proposed techniques targeting other components. Moreover, P_{AI} does not depend on a particular architecture or workload type.

6.2.4 Power dissipation per thread (P_t)

Power dissipation per thread represents the dynamic power dissipation in some sense. Decided by how efficiently system resources are being used, dynamic power dissipation associates with run-time system management, system architecture, workload characteristics, and so on. Consequently, various factors affect power dissipation per thread. For instance, database applications exhibiting high memory and I/O utilization have distinct features from computation intensive applications in terms of dynamic power. Another example is low-power electronic devices execute specific types of tasks more efficiently compared to their counterparts. Measuring per thread power on a multi-core processor is challenging; therefore, power models are

used instead of hardware measuring. We developed SPAN [137] to model power dissipation at per function per thread level.

Clock gating is one of the most widely used techniques [142]. By disabling part of the circuits so that they do not have to switch states, clock gating saves dynamic power. Workload characteristics mainly determine the effects of clock-gating. General purpose processors, although have more computation power, usually cannot satisfy low power features. Application-specific integrated circuit (ASIC) is much more energy efficient, for certain types of tasks [54]. For example, most of the latest smart phone platforms have GPU units, which perform graphic jobs more efficiently. In this sense, it reduces P_t required to finish certain tasks.

Another principle to reduce dynamic power is to put devices or components into lower power modes if they are not in use. DVFS allows run-time adjustment of power dissipation of a CPU. According to the equation $P = CV^2F$, reducing voltage and frequency has a cubic effect on power dissipation. However, one of the disadvantages of using DVFS is that it extends execution time T. In this sense, the overall effects of DVFS on $\mathscr{E}$ is uncertain. Normally, because of the existence of static power, extending workload execution time reduces $\mathscr{E}$ even though the average power decreases. The key point is to apply DVFS on applications properly. Isci *et al.* propose a run-time prediction model to identify program phases that are less CPU-bounded [66]. Afterwards, DVFS can be applied to these phases with limited performance loss. Hence, the extended T value can be controlled during this process. The same idea can be applied to similar scenarios. For example, as far as I/O bounded applications are concerned, DVFS effectively improves $\mathscr{E}$. If CPU-bounded applications are the major targets, I/O devices can be safely put into deeper D states.

Other than frequency, workload characteristics also contribute to the per thread power to a certain extend. Activities on different components of a system determine the total amount of power dissipation at that moment. For example, IPC values are highly related to CPU power [137]. Some applications suffer from high last level cache (LLC) misses, which leads

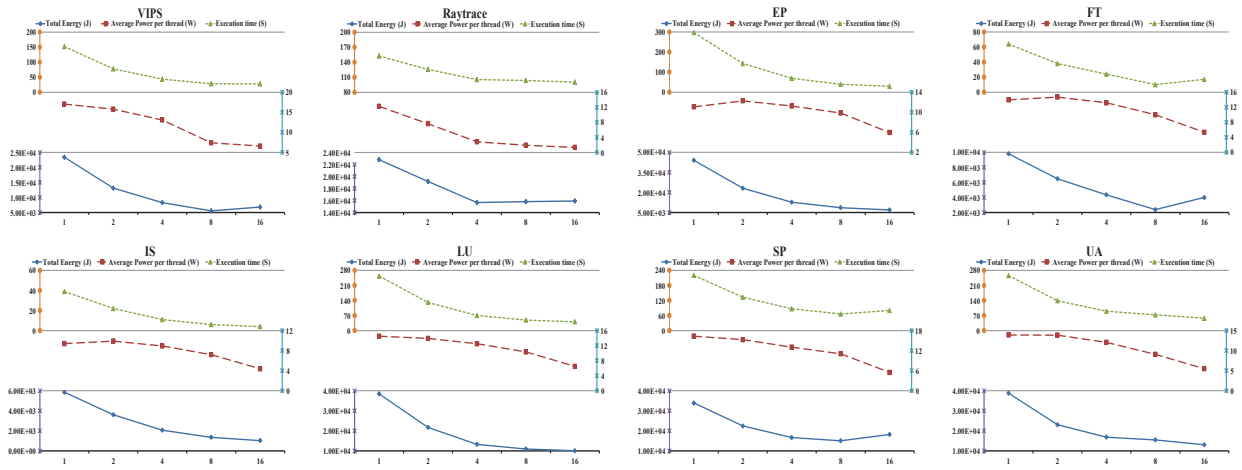


Figure 6.1: Execution time, average power dissipation per thread, and total energy consumption of NPB and PARSEC benchmark; The X-axis represents total number of threads (C); (from bottom up)The first Y-axis depicts energy consumption in Joules; The second Y-axis is for average total power dissipation per thread (P_t); The third Y-axis shows total execution time (T); P_{AI} stays around 137W. CPU frequency is set to 2.4GHz.

to high memory power dissipation and low $\mathscr{E}$. Either by altering the implementations or algorithms, P_t can be controlled. However, optimization techniques used at this level sometimes fall into the same category of performance optimization.

6.3 Case Study

In this section, we use three case studies to illustrate the usefulness and effectiveness of CPT: 1) the effect of concurrency (C); 2) the impact of thread mapping; and 3) the influence of DVFS.

Experiment setup We conduct the experiments on an Intel Xeon E5620 server. The specifications are listed in Table 6.1. There is a total of eight frequencies available, with the maximum of 2400MHz and minimum of 1600MHz. We use the NPB benchmark suite with OMP implementation and PARSEC benchmark to demonstrate the idea of CPT. In order to measure the energy consumption of the workload on the system, we connect a power measurement device, Watt's Up Pro [95], between the power outlet and the server. Watt's Up Pro is able to record power dissipation of the entire system at a frequency of 1Hz. It is connected to the system with

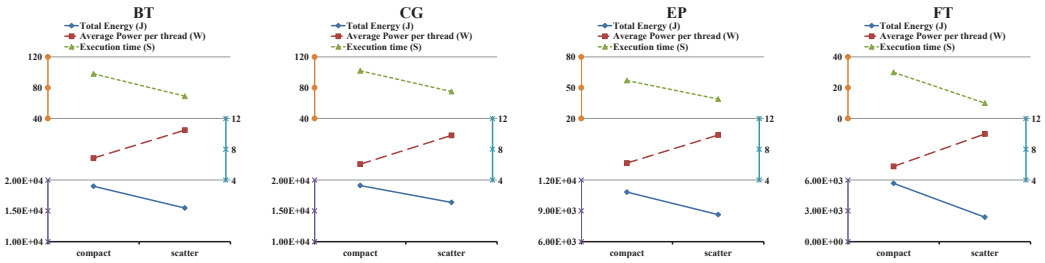


Figure 6.2: Execution time, average power dissipation per thread, and total energy consumption of 8 thread version of NPB-OMP benchmark; The X-axis represents different thread mapping strategy; (from bottom up)The first Y-axis depicts energy consumption in Joules; The second Y-axis is for average total power dissipation per thread (P_t); The third Y-axis shows total execution time (T); P_{AI} is relative stable.

System component	Configuration
CPU	Intel Xeon E5620
Microarchitecture	Nehalem
Processor core	Westmere-EP
L1 cache	4 × 32KB I cache 4 × 32KB D cache
L2 cache	4 × 256KB
L3 cache	12MB
Frequency	2400MHz
Number of sockets	2
Num of cores per chip	4
Num of threads per chip	8
Total num of threads	16
Kernel version	Linux 2.6.31

Table 6.1: System specification.

a serial port. Watt's Up averages power measurement within one second intervals, so that it is safe to use power readings and execution time to calculate total energy consumption.

Case Study 1: Concurrency: We show the effects of concurrency on other factors and $\mathscr{E}$. Firgure 6.1 demonstrates the effects of concurrency on the average power dissipation, workload execution time, and total energy consumption. As discussed, increasing concurrency level properly can reduce execution time. The specific speedup factor varies among different workloads. EP, IS, LU, and UA benchmarks are shown to be most affected by the concurrency level. However, most of them suffer from the diminishing of speedup. Most benchmarks in this category show less speedup if the number of threads utilized is equal or greater than four.

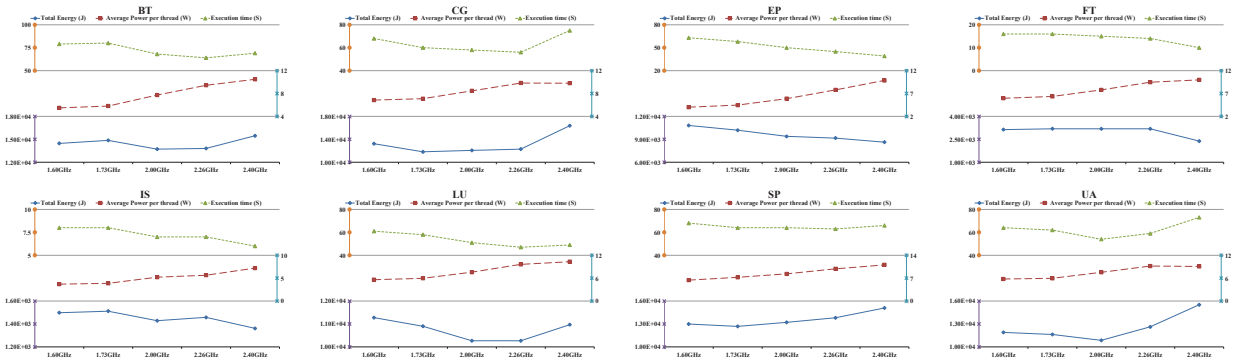


Figure 6.3: Execution time, average power dissipation per thread, and total energy consumption of 8 thread version of NPB-OMP benchmark; The X-axis represents different CPU frequencies; (from bottom up)The first Y-axis depicts energy consumption in Joules; The second Y-axis is for average total power dissipation per thread (P_t); The third Y-axis shows total execution time (T); P_{AI} is relative stable and stays around 137W for all the cases. Scatter is used as the thread mapping scheme.

Although FT, VIPS, and SP benchmarks benefit from concurrency, execution time is no longer monotonically related to the number of threads. The execution time increases if all 16 logical cores are employed. For the Raytrace from PARSEC, it reaches optimal energy efficiency when only four cores are used. This is a typical case to consider because using more threads does not improve energy efficiency for Raytrace. It is because resource contention and serial portion of the workload become dominant factors rather than computation needs.

Average power dissipation per thread, P_t, decreases generally if more cores are involved in the computation. The exceptions are EP, FT, and IS benchmarks when two cores are deployed. The reason is probably because more function units on the chip are operational if two cores are used. As a result, techniques such as clock gating is no longer in use. It is worth noticing that the total average power dissipation, which is the sum of P_{AI} and $C \times P_t$, increases monotonically as more cores are used even though the value of P_t drops in most cases. The extra energy consumption due the difference in power dissipation does not sabotage the overall energy efficiency because of the speedup. However, as speedup diminishes, this part of energy consumption affects the overall energy efficiency.

In this set of experiments, the optimal configuration that minimizes execution time matches the configuration generates most energy efficiency. The reason is because P_{AI} occupies a great portion in the total power dissipation even if all 16 logical cores are deployed; as a result, $P_{AI} \times T$ contributes a large portion to the total energy consumption. Because of the existence of a large amount of static power due to the smaller transistor size, "race to idle" plays a vital role to achieve the most efficiency configuration. In addition, it is worth noting that increasing concurrency level always generates positive speedup results in the tested benchmarks if no simultaneous multi-threading (SMT) is considered, which indicates the improved balance between multi-core CPU and memory subsystem in terms of speed. The improved memory performance mainly can be attributed to NUMA architecture. Moreover, although SMT can be effective in most of cases, its usage depends on data demands of the workload.

Case Study 2: Thread Mapping: Maintaining a constant CPU frequency and workload concurrency level, the organization of threads on a system also affects total energy efficiency. This technique is known as thread mapping. Compact thread mapping means that the threads are allocated to as less processors as possible. This approach reduces data access latency since sibling threads are sharing the same off-chip cache. Scatter scheme assigns thread evenly to each processor, which reduces off-chip resource contention, such as LLC. Figure 6.2 shows the effects of thread mapping on energy consumption when eight threads are used. All benchmarks exhibit similar behavior. A system consumes less power by using only one processor. However, execution time is reduced considerably if two processors are deployed together. On average, there is 33% execution time reduction. The combined result is that a total of 22% energy saving can be achieved if scatter thread mapping is used. Speedup probably comes from fully utilized off-chip resources from two sockets. The results do not necessarily show that a scatter mapping scheme outperforms a compact mapping scheme in terms of energy efficiency in all cases. For example, if only four threads are deployed, sometimes a compact mapping scheme consumes less energy. As Figure 6.4 shows, a compact thread mapping scheme achieves better energy

efficiency for BT benchmark (SMT is not used). The major advantage of using a scatter policy is that an additional LLC can be involved in the computing. In other words, compact policy can be efficient if the working set size is small enough, in which case, the compact policy will consume less power (P_t) with a limited amount of performance loss (T) or even performance gain. Because of this characteristics, compact policies can be used for power capping as well. We discuss it in more details in the next case study.

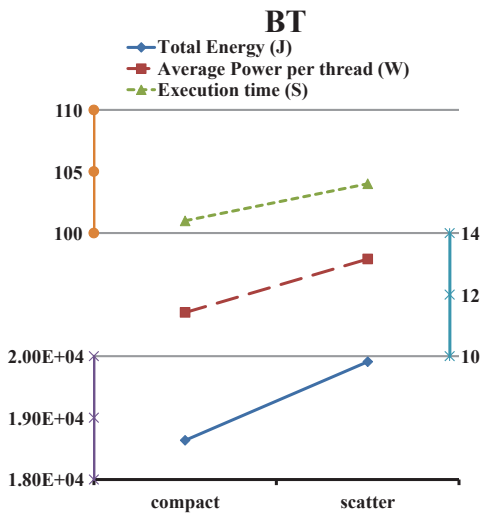


Figure 6.4: Execution time, average power dissipation per thread, and total energy consumption of 4 thread version BT benchmark with different thread mapping strategies; P_{AI} is relative stable and stays around 137W for all the cases. CPU frequency is set to be 2.4GHz.

Case Study 3: DVFS As we discussed in Section 6.2.2, DVFS is an effective way to reduce P_t. However, execution time increases because of the compromised computation capability. Figure 6.3 shows the effects of tuning the CPU frequencies for different benchmarks. It is obvious that the most energy efficient frequency is depend on the particular workload. For example, at 2.00 GHz, BT, LU, and UA benchmark achieve most energy efficiency among all the different frequencies. While, for SP and CG, it is 1.73GHz. In addition, a finer tuning can be made for each benchmark to achieve better energy efficiency. We pick IS benchmark as an example to show the effects. We use Intel Core 2 Quad 8200 as a experiment platform in this study to measurement CPU power dissipation directly from the power supply. We use

SPAN [137] to record different functions activities in IS benchmark. There are three major steps in IS.A: `create_seq()`, `rank()`, and `full_verify()`. In this example, the `rank()` function, which produces approximate 0.15 Instructions Per Cycle (IPC), is less CPU-bound during its execution (IPC values are broadly used as CPU power model input [137]). For example, we are able to use DVFS to scale down CPU frequency from 2.34GHz to 2.00GHz during the execution of `rank()`. As Figure 6.5 shows, we achieved 24% energy saving with 3% performance loss. On the contrary, 10% energy saving is achieved with 10% performance loss if we scale down CPU frequency during the execution of `create_seq()`, which has higher IPC values during its execution (around 0.6). In addition, we observe that multiplication operations are intensively used in the source code of `create_seq()`, while `rank()` function mainly contains branch-prediction and data movement operations. Hence, tuning P_t using DVFS will effect both execution time and power dissipation. In order to improve $\mathscr{E}$, such a technique needs to be carefully applied.

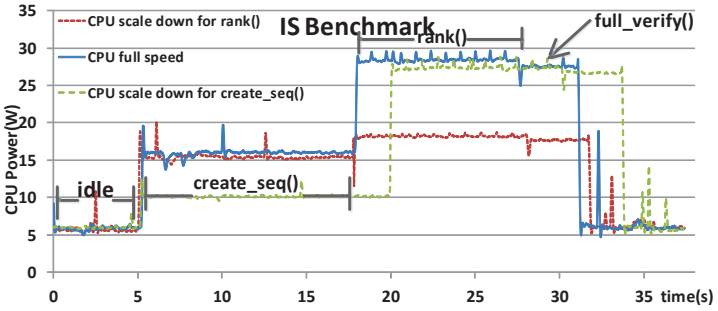


Figure 6.5: Power dissipation of IS on a Intel Core2 Quad 8200 processor with different DVFS setting.

Given a cap power value [83], using thread mapping and/or DVFS can control the power dissipation of a system. The CPT model also can be used to demonstrate this situation. Since $P_{AI} + C \times P_t$ is fixed (due to the cap), the system configuration that generates highest performance provides maximum energy efficiency. The specific configuration, however, depends on the workload and the system. Figure 6.6 illustrates this scenario. Although either applying the compact mapping or a slower CPU speed reduces power dissipation, the performance loss

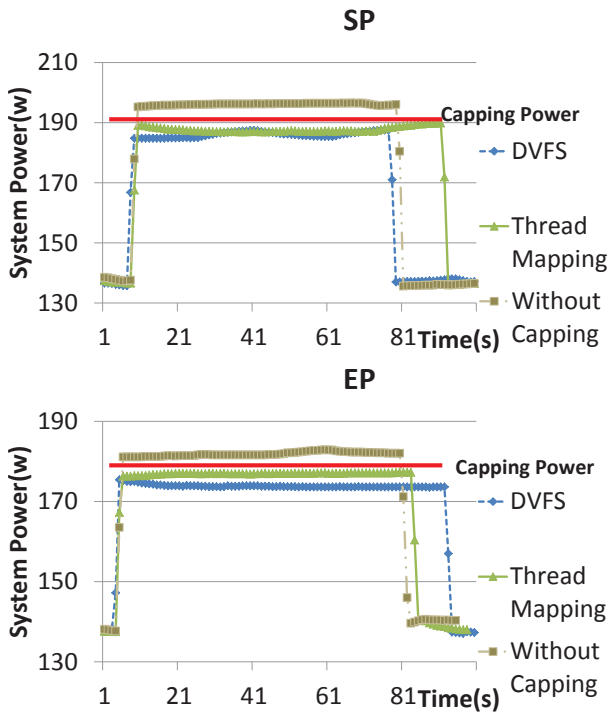


Figure 6.6: Power dissipation and execution time using different power capping techniques.

introduced by DVFS is less for SP benchmark; while it is the opposite situation for EP benchmark. The advantage of using compact threading mapping includes allocating the data to a relative closer cache, other than the remote cache. A recent study [118] shows that memory performance can be affected by DVFS, which should be analyzed based on various computer systems. In other words, either using a different thread mapping strategy or DVFS, the off-chip data access can be affected. However, compact thread mapping produces a more energy efficient result if the workload phases tend to be computation intensive. EP benchmark can be considered as an extreme case. Although the DVFS strategy reduces off-chip data access bandwidth as well, it utilizes all the LLC from both processors, which results in a higher performance gain for a certain set of benchmarks.

6.4 Related work

In the data centers, regarding the effectiveness of the power usage, Green Grid first officially introduces the equation:
$PUE = Total\ Facility\ Power / IT\ Equipment\ Power$ [111]. The formula is widely used to evaluate the efficiency of a whole data center design. The proposed CPT model is complementary with PUE in the sense that CPT emphasizes useful work produced by a system.

In-depth understanding of software power dissipation becomes one of the major considerations when designing power-aware systems. Ge *et al.*, propose PowerPack [46] to generate component level power profiles. This approach targets on cluster level. PowerPack provides APIs to synchronize external power measurement and function execution of the target application. However, manual instrumentation is inconvenient for large scale applications. Hänig *et. al*, propose SEEP [58], which uses symbolic execution to explore possible code paths and entities in a program and to generate energy profiles for specific target platforms. Instruction level energy profiles are needed for each platform in advance in order to generate energy profiles for a program.

6.5 Summary

In this chapter, we propose a general CPT model to analyze the system energy efficiency for a given workload. Most techniques on the market can be categorized as altering parameters in the proposed model. We show three case studies to illustrate how to use CPT model to analyze different techniques. In practice, each technique proposed can be examined from the aspects mentioned in Section 6.2. Energy efficiency is closely related to the system architecture, workload characteristics, and system configurations. We expect the CPT model helps to identify the bottleneck of existing systems and serves as guidance for future energy-efficient system designs.

Based on the power model proposed in the previous chapters, each item in CPT model is measurable without hardware instrumentation. While CPT model only describes the factors

that affect energy efficiency, in the next chapter, we closely estimate and optimize each one of them in a production system in order to achieve better energy efficiency for a given workload.

CHAPTER 7

ENERGY-EFFICIENT SYSTEM CONFIGURATION PREDICTION

Based on the proposed CPT model in the last chapter, we analyze various parallel workload. Our goal in this chapter is to achieve the energy efficiency of the whole system while executing such workload. While most existing works concentrate on either static analysis of the workload or run-time predication results, in this chapter, we present a hybrid two-step method that utilizes concurrency levels and DVFS settings to achieve the energy efficiency configuration for a workload. Particularly, we employ the profiling tools that is proposed in the previous chapter to estimate the power dissipation. Specifically, we present models to estimate the speedup and power dissipation of a parallel program for different system configurations, such as the number of cores and the thread mapping strategy. The second step involves using DVFS to adjust voltage/frequency at run-time when applying the configuration obtained from the first step to further reduce energy consumption.

7.1 Introduction

Modern computer systems are designed to balance performance and energy consumption, especially in HPC systems. Several run-time factors, such as concurrency levels, thread mapping strategies, and dynamic voltage and frequency scaling (DVFS) should be considered in order to achieve optimal energy efficiency for a workload. Selecting appropriate run-time factors, however, is one of the most challenging tasks because the run-time factors are architecture-specific and workload-specific.

The focus of computing has shifted from performance-centered to energy efficiency. As a result, energy-efficient techniques have been adopted across different layers in almost every system, from single chips to large data centers [94, 120]. Power dissipation and energy

consumption are priority concerns when designing computer systems, especially in the High Performance Computing (HPC) field. A recent article suggests that the benefits of the multi-core architecture diminishes as the power constraint on a chip rises [38].

Generally, there are two major factors that affect energy consumption for a specific workload: execution time and average power dissipation. Speedup models are used to describe the benefits introduced by parallel implementations in terms of execution time, while power models are used to estimate power dissipation of a workload. Energy efficiency can be defined as the workload over the required energy, which in turn is equal to $\frac{W}{(P_{AI}+\sum_i^C P_t)T}$ [138], where C is concurrency level, P_{AI} and P_t denote active idle power of a system and average power dissipation of each thread, respectively, and T is execution time. A concurrency level with a thread mapping strategy is referred to a configuration in the rest of the chapter.

In-depth analysis of these three factors, C, P, and T, is necessary to achieve better energy efficiency. For example, a speedup model is usually used to quantify the benefits introduced by parallel computing in terms of execution time [78]. Higher concurrency levels, however, affect power dissipation (P) because not only additional computing units are activated but also the power dissipation of common components on a chip will be shared by more cores. An analytical model is needed to understand the energy efficiency of a workload in a multi-core computing scenario. While allocating a workload to multiple CPUs is an effective way to reduce computation energy consumption, DVFS is usually used to explore slacks during execution to save extra power dissipation [62].

Workload characteristics and micro-architectures have major influence on the three factors. The speedup factor of a workload is closely related to the serial portion of different programs [44, 21]. In addition, memory boundedness affects the scalability of a workload in the sense that an individual thread or a process competes for off-chip resources with other threads so that concurrency hazards, such as false sharing, might occur [36]. That information cannot be exposed without run-time profiling. On the other side of the spectrum, modern computer

systems deploy different mechanisms to improve memory performance. For example, Intel processors use Quickpath technology [151] as an implementation of Non-Uniform Memory Access (NUMA) architecture.

An empirical model is used to predict the configuration, and voltage/frequency levels of a workload [30, 98] by using Performance Monitoring Counters (PMCs). However, one of the major drawbacks of using an empirical model is architecture dependency, which requires different sets of PMCs to be used for different architectures. Ge *et al.* propose an energy-performance estimation using an analytical model [45]. However, the model mainly analyzes the behavior of a multi-core based power aware system by case studies. No prediction is used to select the appropriate configuration for each workload.

In this chapter, we propose an approach to predict the appropriate configuration of a workload for energy efficiency purposes. First, we propose an analytical speedup model that utilizes PMCs to predict potential speedup of various configurations from two threads execution information. The collected information is used to build the power estimation of various concurrency levels. Once the optimal concurrency level is selected, we apply a run-time DVFS to select an appropriate frequency for each phase. Our contribution of this chapter includes the following items.

- We generalize the relationship between C, P, and T using mathematical models.
- We propose a model to capture the relationship between C, P, and T in detail. Execution information using two threads is used to predict the energy consumption of different configurations on a specific architecture. A DVFS scheme is selected during the run-time given the predicted concurrency level and thread mapping setting.
- We apply an analytical speedup model to predict the optimal/near-optimal configuration of a parallel workload using architecture details and PMCs information. By using an analytical model, unlike an empirical model, we reserve the applicability of the model

on different architectures.

The rest of the chapter is organized as follows: we start the chapter by introducing our observation between workload concurrency level and execution time/power in Section 7.2; then, we present a two-step prediction model in Section 7.3, followed by evaluation of the prediction model on a Intel Xeon E5620 platform in Section 7.4. Related work is discussed in Section 7.5. Finally, we summarize the chapter in Section 7.6.

7.2 Observation

In a multi-core or many-core system, the scalability and power dissipation of a workload are closely related to the system architecture and workload characteristics. Considering speedup factors, the execution time is infinity if no computation unit is involved. As the core number and thread number increase, execution time drops since more computation power is involved. The lower bound of the execution time, however, is limited by two factors, namely the serial portion of the workload and the off-chip resources. As a result, the execution time of a workload approaches its lower bound and even slowly rises as a system allocating more cores to the workload. On the other hand, a system power dissipation increases as the workload occupies more cores. The system consumes idle power if no computation power is invoked. Although the system power dissipation increases, it is bounded by the Thermal Design Power (TDP). As a result, the speedup bound and power bound can be modeled using mathematical equations.

Specifically, we model the speedup and power dissipation as a function of the number of threads utilized as Equation 7.1 and 7.2, respectively. In particularly, we derive Equation 7.2 from the Logistic Function, which has a maximum value β_1 and exponential growth. This scenario corresponds to the fact that the system power is bounded by the design. $\frac{\beta_1}{1+\exp^{\beta_2}}$ represents the system idle power when there is no cores engaged for the workload.

$$T = f(C) = \alpha_1 + \alpha_2 \times C^{\alpha_3} + \alpha_4 \times C^{-\alpha_3} (\alpha_3 > 0) \tag{7.1}$$

$$P = g(C) = \frac{\beta_1}{1 + \exp^{\beta_2 - \beta_3 \times C}} \tag{7.2}$$

Figure 7.1 shows the model fitting results for NPB and PARSEC benchmarks. The solid line represents the proposed model while the dots are actual measurements. It is easy to observe that all the benchmarks present a similar pattern, which can be captured by the mathematical models. As the number of threads increases, the workload speedup is limited by the off-chip resources such as cache and memory. On the other hand, the power dissipation, starting from idle power when no thread is running, grows to meet the maximum bound. The mathematical models carefully capture the features of concurrency level and its relationship to the power and speedup. Particularly, the execution time and power dissipation of raytrace benchmark fall into a small range because of the large serial portion of the benchmark. Our model is able to capture this unusual case as well.

We derive models to estimate speedup and power dissipation under various configurations for a workload in order to achieve the best energy efficiency and energy delay product.

7.3 Model Derivation

In the first step, we focus on concurrency levels (C), power dissipation of the selected concurrency level (P), and execution time (T). By increasing the concurrency level of an application, the average power dissipation increases while the execution time required to finish the same work shrinks. Usually the speedup benefited from a higher concurrency is substantial so that the total energy consumption drops. However, a large number of threads in a system usually results in a competing situation, especially regarding front bus usage. Although NUMA [22] is proposed to solve this problem, off-chip activities affect speedup dramatically. Thread mapping is another technique that is proposed to reduce contention. However, the effects of thread mapping on energy efficiency are uncertain [57, 24]. Although a compact scheme (allocating threads to as less physical processor as possible) generates less power dis-

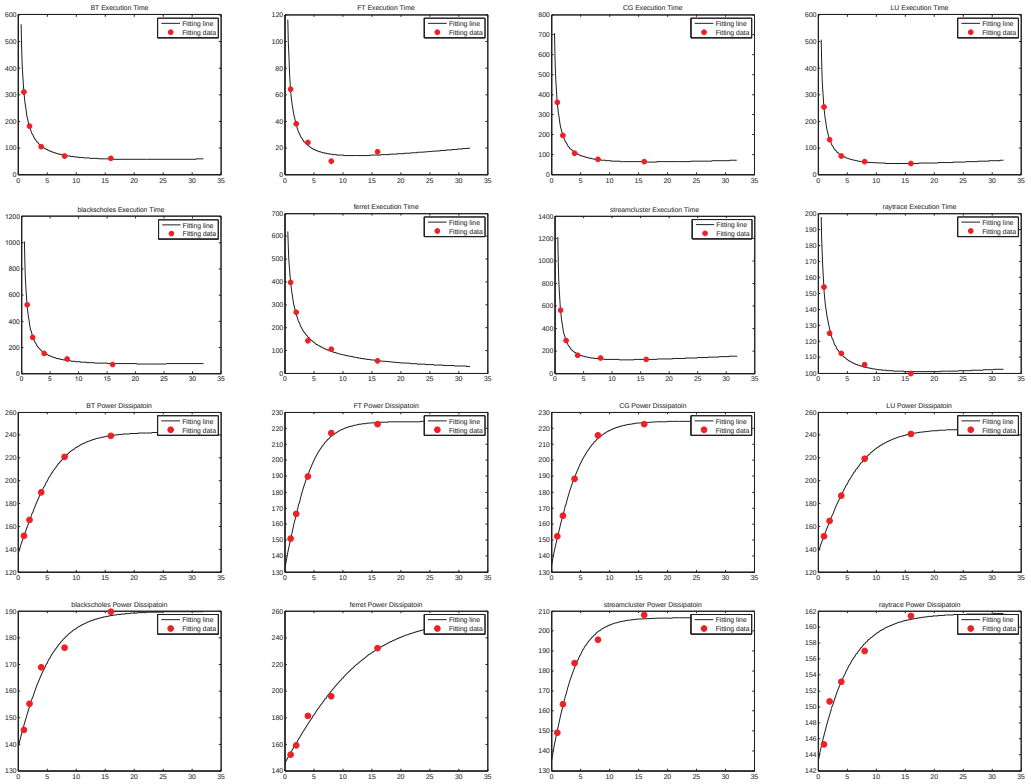


Figure 7.1: The model fitting of speedup and power dissipation of the workload as a function of concurrency level. The benchmarks are BT, FT, CG, LU, blackscholes, ferret, streamcluster, and raytrace, respectively.

sipation, this scheme tends to reduce speedup. Scatter scheme (allocating threads evenly to different physical processors), on the other hand, generates higher power dissipation but alleviates contention.

7.3.1 Analytical Speedup Model

In this section, we derive a speedup model for different configurations. The input for the model is collected from the parallel execution with two threads. The output of the model is the speedup factor when applying different thread mapping strategies and concurrency levels. The maximum CPU frequency is always applied in this section. We discuss the model based on the following assumptions and considerations:

- We assume the workload is allocated to a dedicated node and the workload executed

directly on the node without virtual machines.

- We measure the workload using two threads and executed on two processors. The collected data are the model inputs.

- We apply a simplified memory model that only uses Last Level Cache (LLC) references and misses to quantify the off-chip data accesses.

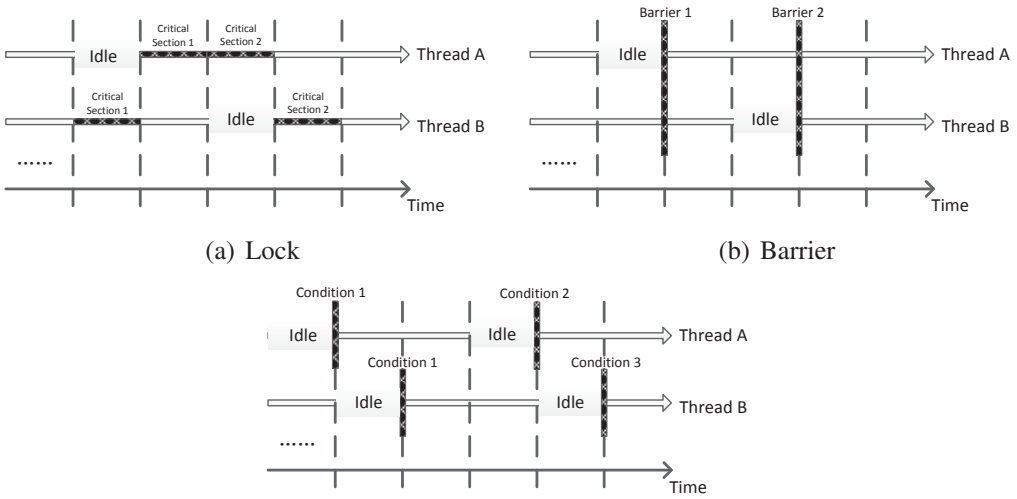
We assume each workload W is composed of two major parts. One is serial portion, W^{serial}, and the other is parallel portion, $W^{parallel}$.

$$W = W^{serial} + W^{parallel} \tag{7.3}$$

3.11 Serial Portion

Serial portion of the workload comprises synchronization cost and workload allocation steps; while the former occupies the major portion of a whole W^{serial}. Briefly, synchronization costs include locks [43], barriers, and condition variables. Currently, there is no effective prediction method, as far as we know, to estimate the exact amount of the serial portion of a workload without profiling it in a real world machine. For example, in a recent study [69], Joao *et al.* designed a new instruction that tracks the amount of cycles elapsed while executing MWAIT instruction in order to identify the serial portion of a parallel program.

Our goal is to predict the synchronization cost of a workload for different system configurations given the information of executing the workload using two threads. Figure 7.2 shows different synchronization strategies that can be deployed in a parallel program. Normally threads need to pause and wait for resources or conditions in those synchronization phases. Specifically, threads monitor the data located in some memory addresses to detect any changes. During a monitoring period, a CPU core usually enters the idle state. In order to predict the synchronization cost, our idea is intuitive. The time duration that each thread waits



(a) Lock (b) Barrier

(c) Condition

Figure 7.2: Synchronization

for each other in the two threads execution could be similar to the amount of time each thread waits for others if n threads are used. For example, data in a critical section protected by locks needs to be processed during the execution. It is the same amount of time that is required to process the data no matter how many threads are utilized because of mutual exclusion. We list the average idle time for each thread using different number of threads of two PARSEC benchmarks with native input data, where i denotes the number of processors and j denotes the total number of threads in Table 7.1. *blackscholes* is mainly synchronized by barriers; while *freqmine* uses locks to protect critical sections [14]. It is easy to observe that the average time each thread spending in the idle mode is approximately the same for different configurations. We summarize the relationship in Equation 7.4.

$$W_{i,j}^{serial} = W_{2,2} \times (i \times j - 1) \tag{7.4}$$

3.12 Parallel Portion

Benchmark	i	j	Avg. idle time per core (s)
blackholes	1	2	28.83
	1	4	29.23
	2	2	30.17
	2	4	29.28
	2	8	28.89
freqmine	1	2	4.33
	1	4	4.41
	2	2	4.31
	2	4	4.18
	2	8	5.30

Table 7.1: Average idle time for each core using different configurations.

The total amount of workload can be expressed in several ways based on each stage in a pipeline, for instance, $W^{parallel} = total\_issue\_cycles + total\_issue\_stall\_cycle$ or $W^{parallel} = total\_active\_execute\_cycles + total\_execute\_stall\_cycle$. At the issue stage, stalls mainly stem from instruction cache misses, resource (such as registers) unavailability and other activities, which result in no instruction being allocated to the Reservation Station (RS). These activities do not exhibit a strong relationship to the selected concurrency level. While $total\_execute\_stall$ measures the delay that is taken to prepare the data for instruction execution.

Part of $W^{parallel}$ is independent of the concurrency level and the thread mapping strategy. Computation load belongs to this part. We denote this part as C. Another part changes significantly mainly because of memory subsystems, which is denoted as M. The unit used in the model is cycles.

$$W^{parallel} = C + M \tag{7.5}$$

The computation load, C, once a different concurrency level and thread mapping strategy are applied, can be evenly distributed to the newly assigned threads. The total workload becomes $W_{i,j}^{parallel}$ and the computation load becomes $C_{i,j}$. The execution time of $C_{i,j}$ part can be reduced to $\frac{2 \times C_{2,2}}{j}$ for each thread. Basically, we use $C_{2,2}$ to estimate the portion of the workload

Benchmark	i	j	$Eexcuted\_Cycles$
$bt.A$	1	1	205174317772
	1	2	205842270932
	1	4	210569866603
	2	2	208036560725
	2	4	210757446738
	2	8	214207223107
$ft.B$	1	1	168308642166
	1	2	169576463590
	1	4	174047613387
	2	2	173953520506
	2	4	175101409435
	2	8	185063427379

Table 7.2: $C_{i,j}$ of bt.A and ft.B benchmark.

that has been sent to different executing ports on processor units[1] under the circumstance that all the operands of each instruction are ready for execution. $C_{2,2}$ counts the total time spent on these executions. It is worth noticing that thread mapping strategies and different concurrency levels do not affect the computation workload in the analytical model.

We measure bt and ft from NPB benchmark suite to verify the idea. We sample PMC $UPOS\_EXECUTED\_ACTIVE\_CYCYCLES$ (E_Cycles for short), whose event number and umask is 0xB1 and 0x3F respectively, using different configurations. The results are shown in Table 7.2. It is clear that the even though the concurrency level and thread mapping strategy changes, the E_Cycles only varies at most 4% and 9% for bt.A and ft.B, respectively.

Changing configurations affects $M_{i,j}$. $M_{i,j}$ part can be interpreted as the total time used for preparing execution on each unit. Different configurations introduce varying amount of $M_{i,j}$. Even though $M_{i,j}$ is shared by j threads, the total amount of $M_{i,j}$ is related to the specific architecture and is the one that needs to be predicted for each pair of i and j. Table 7.3 illustrates the $M_{i,j}$ part of bt.A and ft.B benchmarks, which is referred as $Stall\_Cycles$. It is calculated by subtracting the $Execute\_Cycles$ from the total number of cycles required to finish the workload.

[1]For example, there are 6 different ports on Intel Microarchitecture Code Name Westmere, where port 0,1, and 5 handle integer arithmetic, SIMD, integer shift, FP multiply and FP divide Uops and Port 2, 3, and 4 handle the load and store Uops[63].

Benchmark	i	j	$Stall\_Cycles$
bt.A	1	1	24536512493
	1	2	27213960868
	1	4	39855366148
	2	2	21026377128
	2	4	33466947535
	2	8	55621254767
ft.B	1	1	18113150861
	1	2	31063082823
	1	4	81392657035
	2	2	23548254140
	2	4	42809740979
	2	8	110260939317

Table 7.3: $M_{i,j}$ of bt.A and ft.B benchmark.

As much as 600% difference can be observed for the benchmark if a different configuration is applied.

The goal of the prediction model lies on predicting the $M_{i,j}$ for a given $M_{2,2}$. One of the most significant resources stalls is off-chip cache accesses. In a NUMA architecture, a 40 cycles to at most 300 cycles penalty can be triggered in order to access an off-chip L3 cache, while accessing memory triggers 60 ns to 100 ns delay [31]. Although the exact number may vary, off-chip data accesses become one of the major root causes of execution stalls. As a result, in order to predict $M_{i,j}$, we record LLC reference and LLC miss rates as the input of the model. The unit of them is counts per sec, which shows the approximate demand of LLC and memory accesses during a fixed interval. The prediction is shown in Equation 7.6, where we denote LLC references and misses as LLC_R and LLC_M respectively. $Stall\_Cycles$ is referred as $Stall$ for short.

$$\begin{aligned} M_{i,j} = &\alpha_{i,j} \times LLC\_R_{2,2} + \beta_{i,j} \times LLC\_M_{2,2} \\ &+ Stall_{2,2} \times \gamma_{i,j} + \varepsilon_{i,j} \end{aligned} \tag{7.6}$$

In Equation 7.5, the percentage of $C_{2,2}$ and $M_{2,2}$ is pre-determined by the workload while in Equation 7.6, the prediction result of $M_{i,j}$ is determined by both the system architecture and the workload itself. Combine Equation 7.5 and 7.6, we obtain Equation 7.7.

$$\begin{aligned} W_{i,j} &= C_{i,j} + M_{i,j} \\ &= \frac{C_{2,2}}{i} + \frac{\alpha_{i,j} \times LLC\_R_{2,2}}{i} \\ &+ \frac{\beta_{i,j} \times LLC\_M_{2,2} + Stall_{2,2} \times \gamma_{i,j} + \varepsilon_{i,j}}{i} \end{aligned} \tag{7.7}$$

In order to obtain $\alpha_{i,j}$ and $\beta_{i,j}$, $\gamma_{i,j}$, and $\varepsilon_{i,j}$, a set of training benchmarks can be used. These parameters are determined by the system architecture, which can be characterized by a set of carefully designed training benchmarks. We modified the memory mountain benchmark [19] by adjusting stride and working set sizes. By changing the stride, we are able to obtain different *LLC_R* and *LLC_M* values. Then, we run the modified memory mountain benchmark using different number of threads and thread mapping strategies. PMCs information is collected during each execution. In addition, system power dissipation information is recorded. A Linear Regression model is used to train the model parameters.

We observed that there are two categories to consider based on the training results. The first category exhibits less data demand that the contention on LLC and memory is limited. The other category shows strong contention by increasing the concurrency level. Two groups are separated by using a pre-defined threshold value of *LLC_R*. The rationale behind this scenario is that before the throughput reaches the limit of LLC and memory bandwidths, the relationship between *LLC_R*, *LLC_M* and $M_{i,j}$ exhibits differently before and after they reach the limits. We utilize the Minimum Absolute Error to determine the threshold in the training set. Four parameters are obtained for each configuration. We set the threshold, δ, based on *LLC_R* on the target platform to distinguish each category. The threshold, δ, is set to 2360000 references/sec.

7.3.2 Power Model

By changing concurrency level and mapping strategy, system power dissipation changes as well. In order to achieve a better energy efficiency configuration, a power model describing power dissipation using different configurations is needed. To be clear, the power dissipation that is referred to in this subsection is average power dissipation.

In order to predict the power dissipation of each configuration, we extend the methodology presented in [137]. The previous work predicts the run-time CPU power on multi-core systems using PMCs as input. However, it requires the run-time data collection. In this case, the model input is execution information using two threads and the expected output is average power dissipation of each different configuration.

In the power model, we assume the CPUs do not enter deeper sleep modes but $C0$. We denote idle power as P^b and dynamic power as P^d. In this chapter, we only consider the dynamic power contributed from the CPU and the memory. There are various works that study about the relationship between CPU power and PMCs [84, 12, 137, 67]. Instruction per Cycle (IPC) or UOPs per cycle shows a strong linear relationship to CPU power dissipation. We continue to use this relationship to estimate power dissipation. $Instruction\_Issued$ is used in the model instead of $Instruction\_Required$ because some instructions are issued and executed but discarded in a pipeline of an OOO execution unit. Regarding off-chip cache and memory power dissipation, LLC_R and LLC_M are introduced to estimate the demand of accesses to LLC cache and memory subsystems. Equation 7.8 shows the overall estimation for dynamic power.

$$\begin{aligned} P_{i,j} &= P^b + P^d \\ &= P^b + IPC_{2,2} \times a_{i,j} + LLC\_R_{2,2} \times b_{i,j} \\ &+ LLC\_M_{2,2} \times c_{i,j} + d_{i,j} \end{aligned} \tag{7.8}$$

The power dissipation of the configuration using i processors and j threads is determined by four newly introduced coefficients, $a_{i,j}$, $b_{i,j}$, $c_{i,j}$, and $d_{i,j}$ The four parameters are determined by system architecture and can be trained for each pair of i and j. However, the IPC, $LLC\_R_{2,2}$, and $LLC\_M_{2,2}$ are related to the workload characteristics.

In order to obtain $a_{i,j}$, $b_{i,j}$, $c_{i,j}$, and $d_{i,j}$, we modified the Memory Mountain Benchmark [19] by adding a computation part after each data item is retrieved. By adjusting the computation load, the stride, and the working set size in the training benchmark, various combination of IPC, LLC_R, and LLC_M values can be obtained. For example, setting workload set size to be limited to fit in L3 cache, configuring the stride value to pass L2 cache, and adding a computation part to occupy a reasonable portion in the whole workload, we can generate high IPC and LLC_R values while maintaining low LLC_M. The training process is similar to the description of speedup model training. Once $a_{i,j}$, $b_{i,j}$, $c_{i,j}$, and $d_{i,j}$ are trained for the target platform, the power model is able to predict the power dissipation of a different configuration.

The total energy consumption of a workload using various configurations can be calculated as follows:

$$E_{i,j} = \int P_{i,j}\,\mathrm{d}t = P_{i,j} \times W_{i,j} \tag{7.9}$$

7.3.3 Run-time DVFS

In the above sections, we describe a static off-line prediction model that uses workload information and system architecture parameters to obtain the speedup factor and power dissipation of different configurations. Applying DVFS [73, 7], we can tune the the power dissipation of CPU at run-time. The CPU frequency is referred as an item in the configuration of a system [30]. However, the proposed method uses DVFS as a run-time knob to tune the results generated from the analytical model described in the previous section because of the following reason: if CPU frequencies are combined with concurrency levels and thread mapping strategies as configurations, the number of configurations needed to be considered in an analytical

model is huge in a modern multi-core system. For example, on an Intel E5620 platform, there are eight different concurrency and thread mapping configurations (do not consider Intel Hyper Threading technology). If eight different frequency levels are considered, the system will generate a total of 64 different configuration settings. The calculation overhead is considerable if we apply it at run-time. Therefore, DVFS is used as the second step to gain further energy savings.

The rationale of tuning DVFS is reducing power dissipation of a system when the workload enters memory bounded phases [66, 65]. The primary goal of this step is to reduce run-time power dissipation at the minimum cost of performance degradation.

The first part of the run-time prediction is to classify each phase of a workload to different categories. The second part is the prediction algorithm. In our approach, each phase can be abstracted using a vector of features denoted as $V = [v_1, v_2, ..., v_n]$. Even for the finest grained phase detection mechanism, each phase is combined with computation and memory accesses. Hence, memory boundedness is related to the percentage of computation and data-accesses in a phase. Moreover, the phase duration usually maintains at the range of 10 ms to 500 ms [31]. In order to approximate the percentage of off-chip data accesses in one phase, we derive a method similar to the one that is used to predict speedup factors: $Stall\_Cycles$ and $Execute\_Cycles$. The percentage of $Stall\_Cycles$ in the total elapsed cycles can represent the memory boundedness in some sense and is one of the items in the feature vector used to identify memory-boundedness. However, off-chip memory accesses are not the only source of $Stall\_Cycles$. In order to compensate this phenomenon, we use LLC references and misses as the second and third item in the feature vector. To summarize, the feature vector contains $\frac{Stall\_Cycles}{Total\_Cycles}$, LLC_R, LLC_M. We use normalized values of the three elements.

Once the feature vector is determined, we use the modified Memory Mountain Benchmark to generate a set of training samples. We record the values of the three items in the feature vector for each training sample along with the frequency that generates the highest power

dissipation deduction over IPC deduction ratio. This way, the training benchmarks can be used to illustrate the maximum power saving with minimum performance degradation. The desired frequency value and the feature vector are collected. Then, all the training cases are grouped into different classes according to its optimal frequency value. At run-time, in order to predict the behavior of phases, we use k-nearest neighbor algorithm ($k = 5$) to determine the current phase. Euclidean distance is used to determine the similarity between the current phase and the stored training phases. In order to predict the next phase, we use a simple last-value prediction algorithm to reduce the run-time overhead. The algorithm predicts the very next phase will be the same as the current phase.

7.4 Evaluation

7.4.1 Implementation

The proposed prediction model is implemented with two parts. The first one is the static prediction model that collects execution information using two threads to predict the desired concurrency level and thread mapping strategy. The other one is a run-time phase detection model that dynamically changes CPU frequencies. The parameters in the model are hard coded in the program. We use the perf tool, which is available from Linux kernel system call, $\_NR\_perf\_event\_open()$ to sample PMCs values. Perf tool becomes available from Linux kernel 2.6.31 [32].

We collect system information for each core, including idle time for each core from PROC file system. Two Architectural Performance Events are sampled at run-time, which include LLC references, and LLC misses. One of the advantages of using architectural performance events is that they behave consistently across different micro-architectures. Two other event counters in the proposed model, $UPOS\_EXECUTED\_ACTIVE\_CYCYCLES$ and $UOPS\_ISSUED.ANY$ are implemented to be architecture-specific. However, these events are supported by most of the recent architectures, which makes our model extensible. To set the configuration after prediction, we utilize $omp\_set\_num\_threads()$ provided by the OpenMp

library. $sched\_setaffinity()$ system call can be used to change different thread mapping strategies given a CPU topology. The second step works as follows: as a workload starts execution, we sample the PMCs at intervals of 500 ms. The collected values are used to predict the next phase of the execution and set the corresponding CPU frequency. The cpufreq subsystem is used to set different DVFS settings. Xeon E5620 supports one frequency for each processor. We disable Hyper-Threading feature. However, prefetching is enabled. The hardware events selected in the model do not count LLC references and misses due to prefetching, while other events, which are $UNC\_L3\_MISS.ANY$ (0x309) and $UNC\_L3\_HIT.ANY$ (0x308), count all the L3 cache accesses.

7.4.2 Experiment setup

We conduct the experiments on an Intel Xeon E5620 server. The specification is listed in Table 7.4. There is a total of eight frequencies available, from 1600MHz to 2400MHz. We use NPB and PARSEC benchmark suites to conduct the evaluation. In order to measure the energy consumption of the workload on the system, we connect a power measurement device, Watt's Up Pro, between the power outlet and the server. Watt's Up Pro is able to record power dissipation of the entire system at a frequency of 1Hz. It is connected to the system with a serial port. Watt's Up averages power measurements inside one second interval, so that it is safe to use power readings and execution time to calculate total energy consumption.

7.4.3 Speedup Model Evaluation

In this section, we evaluate the accuracy of the proposed speedup model. The input of the model is the information collected from the execution using two threads of the workload. The output is the estimated execution time of different configurations.

Figure 7.3 shows the serial portion of a workload in terms of absolute execution time and percentage to the entire execution time using eight threads. All of the tested benchmarks from NPB demonstrate less than 2% serial portion; while PARSEC benchmark suite contains various serial execution phases, ranging from 3% to 87% of total execution time. The serial portion

System component	Configuration
CPU	Intel Xeon E5620
Microarchitecture	Nehalem
Processor core	Westmere-EP
L1 cache	4 × 32KB I cache 4 × 32KB D cache
L2 cache	4 × 256KB
L3 cache	12MB
Frequency	2400MHz
Number of sockets	2
Num of cores per chip	4
Num of threads per chip	8
Total num of threads	16
Kernel version	Linux 2.6.31

Table 7.4: System specification.

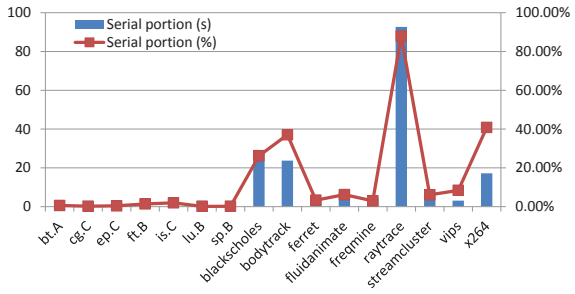


Figure 7.3: Serial portion of the workload

limits the speedup that can be achieved for *raytrace* benchmark, particularly.

Regarding the parallel part, in order to obtain a more accurate prediction, we separate the workload into different categories and train benchmarks for each category. This way, we can obtain the parameters described in Equation 7.6 as shown in Table 7.5. Conf refers to the concurrency level and thread mapping strategy used. Ctgy stands for the categories mentioned in Section 7.3.1. The threshold δ is set to be 2362346 references/sec. γ is used to describe the stall portion. In the case that all four cores are used, namely configuration (1,4) and (2,8), we can observe that the serial setting does not contribute much to the predicted results as other configurations, such as (1,2). As the configuration becomes complicated, the

Conf	Ctgy	α	β	γ	ε
(1,2)	1	-1422.02	16638.10	1.15	-3.37E+10
	2	-1362.78	17921.22	1.02	-2.36E+10
(1,4)	1	118181.23	-99645.46	0.35	6.80E+10
	2	-7743.71	179178.05	-0.35	-3.64E+11
(2,2)	1	6327.38	-6849.06	0.95	4.30E+09
	2	-921.27	22574.88	0.72	-4.80E+10
(2,4)	1	55261.26	-46362.25	0.70	3.28E+10
	2	-5828.06	130347.17	-0.11	-2.78E+11
(2,8)	1	186966.68	-149787.96	0.11	8.73E+10
	2	-7333.23	154658.47	0.42	-2.68E+11

Table 7.5: Parameters obtained for the speedup model.

architectural meaning of each parameter diminishes. The reason is because our model only samples the activities of a small portion of the whole system to maintain its applicability. In addition, there are only at most four performance event registers on the state of the art micro-architectures [63], which means only four events can be simultaneously sampled without multiplexing counter registers. Our model uses all four counters, namely LLC references, LLC misses, $UPOS\_EXECUTED\_ACTIVE\_CYCYCLES$, and $UOPS\_ISSUED.ANY$. The total UnHalted Cycles are calculated from execution time and CPU frequencies.

Table 7.6 illustrates the percentage of stall cycles in the two thread execution of each benchmark. The percentage of stalls is one of the workload characteristics we discussed in Section 7.3.1. If the percentage of stalls is less compared with the computation part, there is more chance that the benchmark speedup is linear. It is easy to observe that both bt.A, ft.B, ep.C, and lu.B exhibit only small percentage of stall cycles in the whole execution. The speedup of all the benchmarks can be observed from Figure 7.4 and Figure 7.5. At the configuration where all eight cores are used, ep.C reaches highest speedup factor, which is 7.60, followed by is.C, which shows 6.45. ft.B and lu.B, however, only speedup 5.63 and 5.45, respectively, if all the eight cores are used. The reason is because these two benchmarks show higher LLC references and LLC misses rates, which are 12 times and 2.5 times on average compared with ep.C and bt.A. As a result, the speedup is limited even though the percentage of stall is only approximately 20% in the configuration of (2,2).

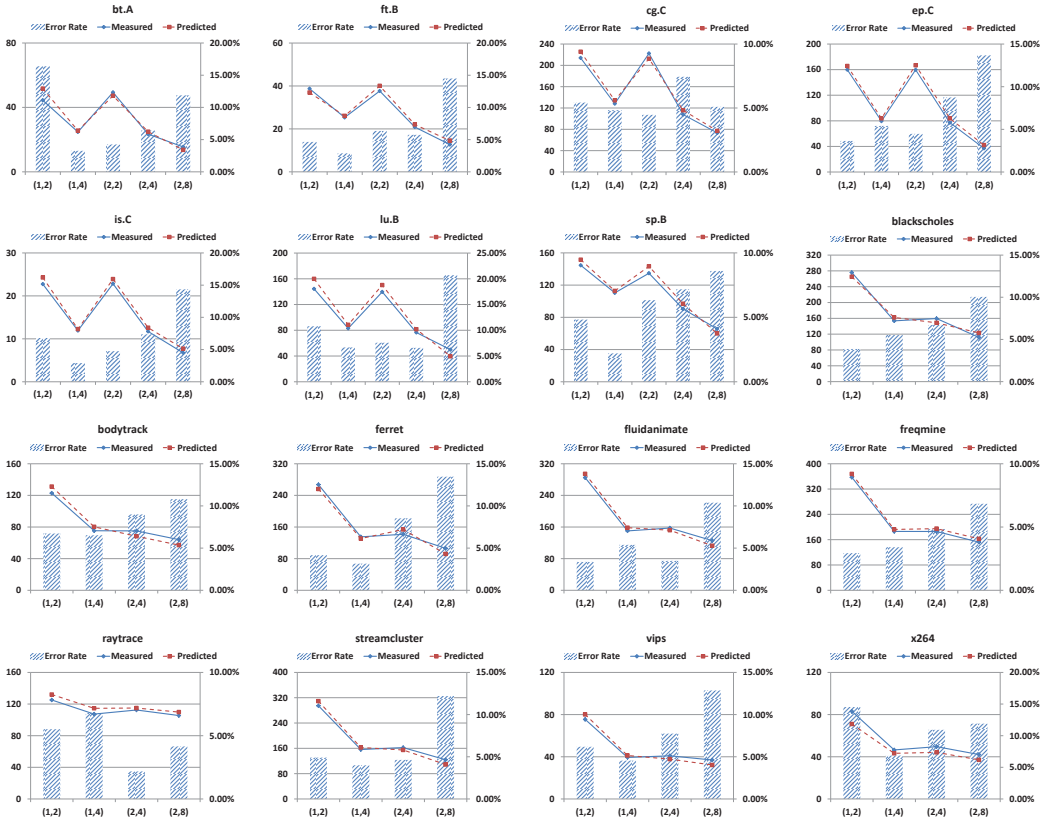


Figure 7.4: Measured execution time vs. predicted execution time. The unit is in second. X-axis represents each configuration. For example, (1,2) stands for one processor and totally two threads are used.

As Figure 7.4 shows, in general the prediction results are more accurate when there are less threads. If all eight cores are used, the average prediction error is 12% compared with the average prediction error in the configuration (1,2) is 8%. The serial portion of *ferret* workload has great variation when all 8 cores are enabled. Specifically, the average waiting time for each thread increases from around 2 seconds to 10 seconds for this particular configuration. The reason is probably because the interplay of condition variables. Regarding *raytrace* benchmark, it has a large portion of serial workload, around 90 seconds, which is the reason why the speedup is restricted.

Benchmark	*Executed_Cycles*	*Stall_Cycles*	Percentage of stall
bt.A	205174317772	24536512493	11%
ft.B	168308642166	18113150861	10%
cg.C	591537183631	493396017764	45%
ep.C	632521811986	168804009044	21%
is.C	66076806330	53651429567	45%
lu.B	587449954695	132446913274	18%
sp.B	496368059294	154413417994	24%

Table 7.6: Percentage of stall in the configuration of (2,2).

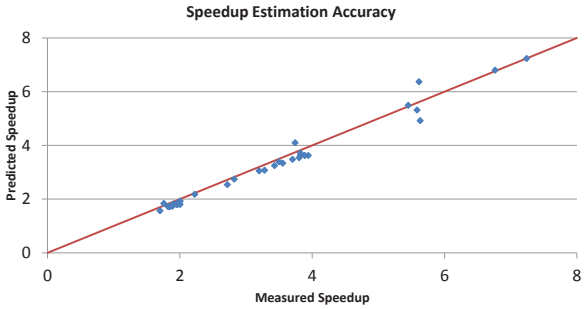


Figure 7.5: The predicted speedup factor using different concurrency levels and thread mapping strategies.

7.4.4 Power Model Evaluation

Power estimation is shown in Figure 7.6. The estimation of power dissipation is a challenging task because of the following reasons: first, the model that we propose does not consider the other components such as motherboards. Those parts are not the major concern of this chapter. However, we measure the whole system power; second, the PMCs used in the model are limited. Basically, only four hardware counter registers are available on Xeon E5620 to be sampled simultaneously. The power model uses four of them, namely LLC references, LLC misses, executed cycles, and upos issued. The average prediction error is 8%. We are able to control the prediction error rate to a relative low level by using only four counters in the prediction model because the idle power for the platform is about 143W, which is a large portion of the whole system. In addition, since serial portion of the workload occupies a certain amount of the the power dissipation for PARSEC benchmarks, they consumes less power compared

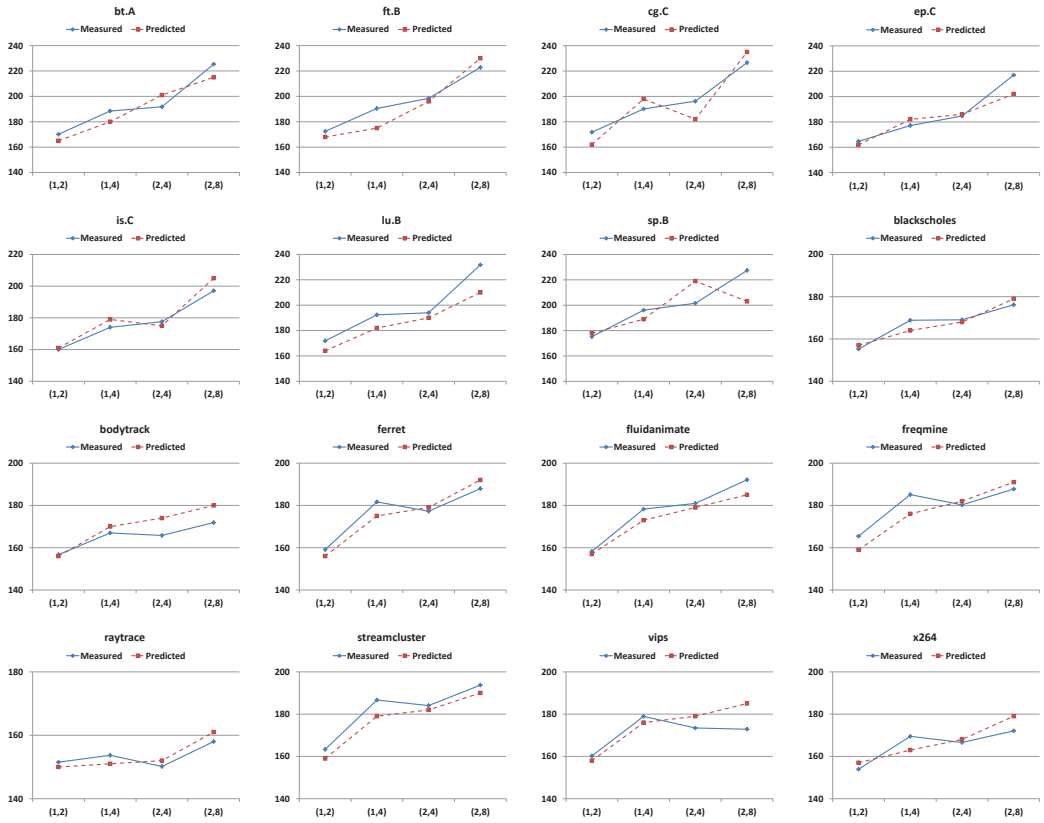


Figure 7.6: Measured power dissipation vs. predicted power dissipation. The unit is in watt. X-axis represents one configuration. For example, (1,2) stands for one processor and totally two threads are used.

with that of NPB benchmarks and are less complicated to predict.

Combining execution time prediction and power dissipation prediction, we are able to calculate the total predicted energy and then determine the most energy efficient configuration. From Figure 7.7, we can observe that our prediction model is able to accurately predict the most energy efficient configuration for each benchmark, which for most of the tested benchmarks is the configuration that uses all eight cores. This result is different from the observation in [30]. The reason is because E5620 uses a shared L3 cache that handles most of the data access of four cores allocated on one socket. This design reduces LLC misses and increases the throughput of the whole system. The extra power dissipation added by this part is negligible compared with the performance gain. One exception is *raytrace* benchmark. Increasing the

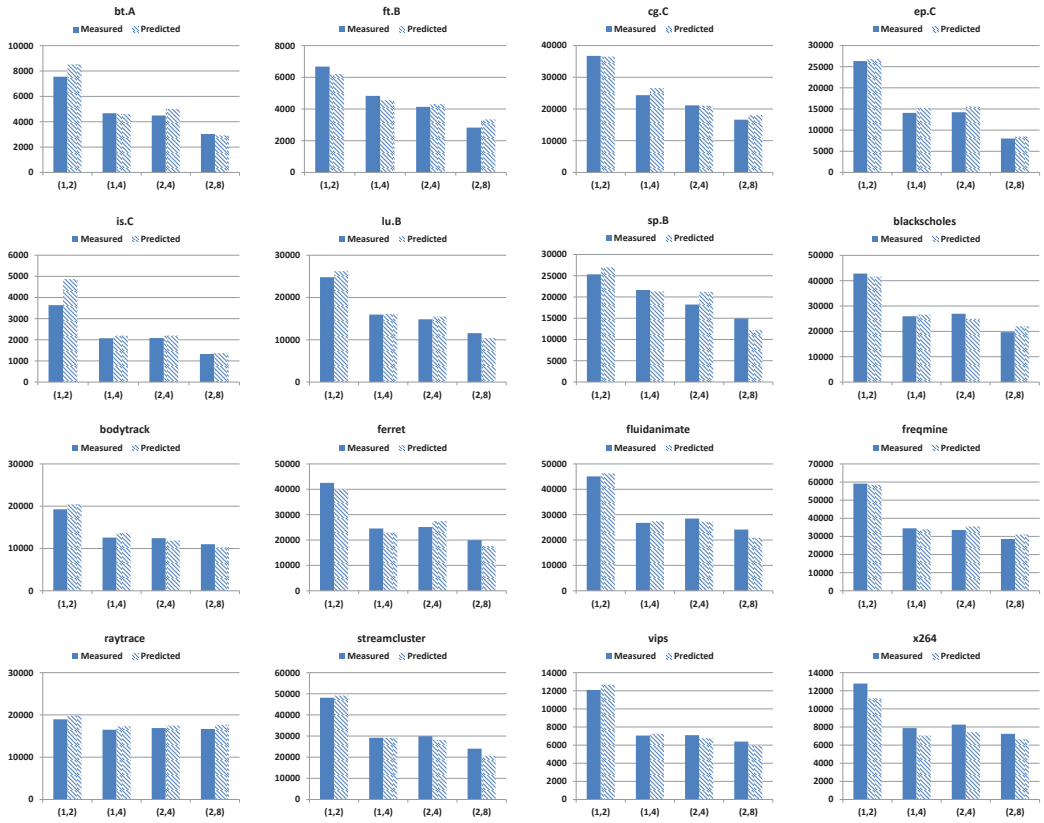


Figure 7.7: Measured energy consumption vs. predicted energy consumption. The unit is in Joule. X-axis represents one configuration. For example, (1,2) stands for one processor and totally two threads are used.

number of threads does not have great impact on the speedup and adds extra energy for the execution because the serial portion of the workload dominates. As a result, using four threads and one processor can produce optimal energy efficiency. However, if Intel Hyper-Threading Technique is enabled and all 16 logical cores are available, we can observe that the benefit diminishes. For FT, SP and vips benchmarks, using more logical cores do not reduce the total energy consumption. But including simultaneous multi-threading in the model is our future work since some of the PMCs can only measure per core events not per thread, which makes the training process obscure.

7.4.5 Run-time DVFS Evaluation

Based on the prediction model in the previous section, we are able to select the optimal configuration for the benchmarks. In the next step, we use run-time PMCs information to predict program phases and select an appropriate frequency level for each phase. Normally, DVFS is only available for a entire processor if only one power domain is designed for the processor.

Figure 7.8 illustrates the additional Energy-Delay Product (EDP) obtained from the the run-time DVFS scheme. The results obtained from the first step uses the maximum CPU frequency. By applying a run-time DVFS scheme we are able to achieve additional energy savings. CG is the most memory-bounded benchmark in the test. It has the largest LLC reference rate and LLC miss rate. The total EDP obtained for CG benchmark is around 24%. On the contrary, no additional saving can be obtained for EP benchmark. *freqmine* benchmark produces maximum EDP saving among PARSEC benchmarks. The reason is because it has a large working set size and frequent data sharing, which produces CPU slacks. The average percentage of EDP saving is 10%. Though there is more CPU idle time during the execution of PARSEC benchmarks, for example, there is as much as 90 second idle time for a CPU when executing *raytrace* benchmark, the energy savings achieved by using DVFS is limited. The reason is because DVFS does not impact power dissipation much during the CPU idle. For instance, the system idle power is around 137W for the tested platform when it either operates at its maximum or minimum frequency. Another method that guides CPUs to enter a deeper sleep mode is needed in order to achieve more energy savings for similar benchmarks.

In order to compare the proposed DVFS scheme and the optimal solution, we apply a brute-force approach to obtain the optimal voltage/frequency for each phase by applying all possible settings. Because we set the sample interval relative large, the number of phases is limited. Figure 7.9 shows the EDP of different benchmarks using the proposed scheme compared with the optimal settings. ep.C benchmark is distinguished from others because the phases are

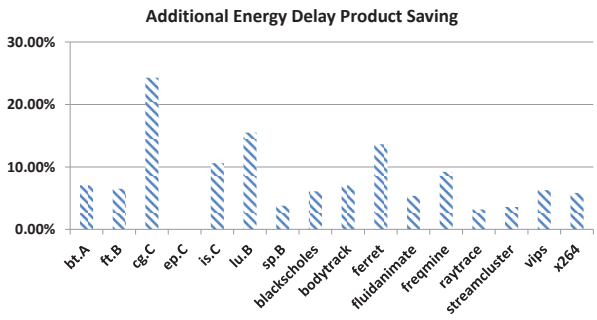


Figure 7.8: Additional EDP using run-time DVFS when the optimal configuration generated from the first step is used. The results are compared to the setting that uses maximum frequency only.

very easy to predict and simply setting the CPU frequency to maximum produces the most energy efficiency solution. Other benchmarks exhibit different characteristics, for example, FT benchmark has various phases and exhibits to be memory-bounded in most of its phases (it has the third largest LLC miss rate, which is 3785337 misses/sec). Phases change frequently in FT benchmark. As a result, FT benchmark spends most of its execution time in the maximum frequency level. CG benchmark, although has the largest LLC reference and LLC miss rate, is supposed to spends most of the time on the second minimum frequency, which is not captured by our proposed model. The prediction model simply sets the frequency of each phase in CG benchmark to minimum frequency all the time. The average gap between the results produced by our approach to the optimal solution is around 5%.

The inaccuracy of the prediction model stems from the following causes. First, the phase prediction model is relatively simple in our implementation. We use a last value prediction algorithm, which performs well if the phase of a workload is stable. Second, one of the assumptions in our two step scheme is that the LLC and memory bandwidths are independent from the CPU operating frequency and the number of threads. In reality, a slight drop in bandwidth occurs on the target platform [118].

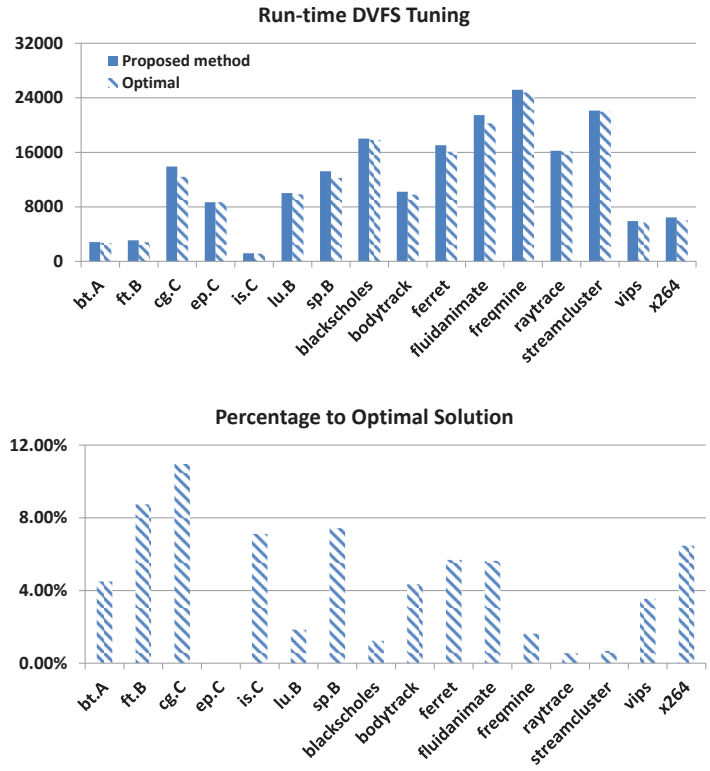


Figure 7.9: Comparing the proposed run-time DVFS adjusting algorithm to the optimal solution in terms of EDP saving. Optimal solution uses an off-line brute-force approach to obtain the optimal frequency for each phase.

7.5 Related work

Different speedup models are proposed to analyze the architecture and program insights of multi-core systems. Kim *et al.* propose an approach that predicts potential speedup from sequential execution [78]. Theoretical analysis of speedup of workloads on modern symmetric and asymmetric is provided in [144]. However, those works do not consider the problem from an energy efficiency perspective. Power aware environment is considered in building speedup models recently. Ge and Cameron [44] propose a power-aware speedup model that is derived from Amdahls Law [4]. The proposed scheme divides parallel workload into two major parts, which include on-chip and off-chip parts. In addition, each one of them can be categorized as two subclasses, one of which is affected by DVFS and the other is not. However, the authors

do not propose methods to predict speedup practically based on the model.

Combining power aware features of a system and parallel workload characteristics, Curtis-Maury *et al.* proposed a multi-dimensional prediction model that uses DCT and DVFS. This work is similar to our approach [30]. However, the method is based on empirical models to predict concurrency level that lack architectural insights, which limit the applicability of the proposed model. For example, different micro-architecture might use different empirical parameters. Ge *et al.* propose an analytical model to analyze the energy efficiency issue using a speedup and power model [45]. The proposed work is verified by case studies. The major difference between the proposed work and [45] is that they focus on analysis while we focus on prediction. Saravanan *et al.* simulates different power features according to the processor characteristics, such as out-of-order execution [115]. However, they did not consider the workloads and their requirements.

Estimating power dissipation using PMCs is one of the most important topics in HPC because the estimated results can be used for peak power control and thermal management. Joseph and Martonosi propose one of the earliest works on estimating power dissipation using Performance Monitoring Counters (PMCs) [72]. Goel *et al.* propose a method to [49] estimate system power dissipation of different architectures. The results show that applying a different set of PMCs according to architectural characteristics produces better estimation accuracy. These methods, however, do not consider the energy efficiency of a workload.

Adjusting DVFS according to program phases is an effective method to reduce energy consumption given a performance metric. One of the most effective methods is scaling down the CPU voltage and frequency during the memory intensive phases. Isci *et al.* introduce a prediction model of a memory intensive phases during a program's execution [66]. Spiliopoulos *et al.* propose a Green Governor that utilizes the slacks in memory-bounded applications to save energy with limited performance loss [127]. The key difference between the proposed approach and the aforementioned approaches is that our approach concerns the scalability of

workloads.

Tuning memory modes to achieve memory savings is also considered to be an effective way. Deng *et al.* propose MemScale which enables DVFS on Memory Controller and DFS on Memory Channels to explore dynamic energy saving [33]. Other than using dynamice tuning, Liu *et al.* changes memory refresh rate of less important data in memory while keeping regular refreshing rate for important data [86] to achieve energy saving. Wu *et al.* propose a bank level controller for memory subsystem that predicts data locality and groups relevant pages togather [141]. However, the hardware techniques on the memory subsystem are not available yet.

7.6 Summary

We achieved the fifth objective in this chapter. Specifically, we propose a practical analytical prediction model that produces energy efficient configurations for parallel workloads. The first step of the model uses execution information of configuration (2,2) and system architecture information to produce the optimal concurrency level and thread mapping strategy by predicting the potential speedup and average power dissipation. DVFS technique is used in the second step to adjust CPU voltage/frequency at run-time to further reduce the energy consumption. NPB OMP and PARSEC benchmark suites are used to evaluate the proposed work. The experimental results based on a Intel E5620 platform shows that the proposed model is able to accurately predict the optimal energy efficiency configuration in the first step. The second step further reduces energy consumption for the configuration obtained from the first step. The average EDP saving is 10%. An off-line optimal solution calculated to compare with the proposed scheme shows that the average extra EDP obtained by optimal solution is within 5%.

CHAPTER 8

CONCLUSION AND FUTURE WORK

In summary, we have successfully achieved all suggested research goals. Tools and models are designed and implemented to fill the needs of power/energy-efficient design in computer systems. Specifically, we proposed the following five items focusing on software workload power analysis and optimization.

1. First, in the study of analyzing component power dissipation of a computer system, we use two experiment platforms of different period (PC_{05} and PC_{10}) to measure the power dissipation of several main components of these two computer systems. The decreasing of static power dissipation and large portion of dynamic CPU power dissipation lead us to the study dynamic power dissipation of software/workload in a computer system.

2. In revealing the dynamic power dissipation in association with workload execution, we present a novel practical power modeling method based on performance monitoring counters (PMCs) by employing one PMCs o recent multicore processors. The proposed model can be used to generate power dissipation information of a workload, which will be used for various tasks. For example, we estimate the power dissipation in CPT using the proposed power model. Based on the model, we design and implement SPAN to map the run-time power dissipation to application functions.

3. In the practice of applying power analyzing model to resource-limited systems, such as embedded systems, we propose a function level power profiling tool, Safari, which produces function level profiling with limited overhead (on average 16% overhead if maximum one sample is collected for each function). It can be used to connect application activities to hardware for energy-efficient design, such as application aware power management and fine-grained scheduling. By designing and implementing Safari, we

are able to achieve similar functionalities in terms of power dissipation as gprof does for performance profiling.

4. In the study of relationship between energy consumption of a workload and the system configurations, we propose a general CPT model to analyze the system energy efficiency for a given workload. We show three case studies to illustrate how to use CPT model to analyze different techniques. The CPT model provides a general abstraction for parallel workload in terms of energy efficiency. Based on the model, we exam the effects of alter each one of them, which leads us to the optimization process in the next item.

5. In design and implement workload-aware energy efficiency strategies, we propose a practical analytical prediction model that produces energy efficient configurations for parallel workloads. The first step of the model uses execution information of configuration (2,2) and system architecture information to produce the optimal concurrency level and thread mapping strategy by predicting the potential speedup and average power dissipation. DVFS technique is used in the second step to adjust CPU voltage/frequency at run-time to further reduce the energy consumption. NPB OMP and PARSEC benchmark suites are used to evaluate the proposed work. The experimental results based on a Xeon E5620 server with NPB and PARSEC benchmark suites show that the model is able to predict the energy efficient configuration accurately for 100% tested benchmarks. An additional 10% EDP saving is obtained by using run-time DVFS on average for the entire system. An off-line optimal solution is used to compare with the proposed scheme. The experimental results using seven parallel benchmarks show that the average extra EDP saved by the optimal solution is within 5%.

In this work, however, we only concentrate on the primary part of the whole system, CPUs. The other components need to be analyzed from different angles. For example, hard disk drive is a typical unsynchronized component, which requires other techniques to analyze its power dissipation. The power dissipation of the workload is closely related to the underling

hardware and system configuration. In addition, the workload in industry are different from those in a research environment. First of all, the complexity of real workload is usually one or two order of magnitude bigger in size. In order to profile program at this scale, we usually utilize divide-and-conquer method to analyze the program part by part. Moreover, modular design is the common approach used to develop large scale programs. The easiest way to use divide-and-conquer is to profile functions in each module separately. This can be achieved using Safari to link to the interested modules. One of the most important difference is that commercial applications usually have "small functions". For instance, the execution of most of functions is less than 100ms. While academic benchmarks usually contains functions with "big body", which makes profiling much easier. In order to implement a function level power analyzer that can be applied to commercial applications, a profiler at least has to leverage the aforementioned points. Regarding the design of power efficiency software, in most cases, the power overhead introduced by performance improvement (improved performance usually uses more system resources, such as prefetch) is not a dominate factor when calculate the overall efficiency. We argue that it might make more sense to tune system component power states based on the software workload.

In the future, it becomes more and more important to provide a off-chip power-resource usage model that models the cache and memory usage. We use benchmarks to estimate the data demand for each workload. Incorporating the characteristics of bandwidth information in the analytical model can improve the applicability of the proposed CPT model. The current work does not consider the Simultaneous Multithreading(SMT) [117]. For example, if Hyper-Threading is enabled on Xeon E5620, the speedup model and power model need to be revisited. In addition, a finer granularity of DVFS tuning is possible to captures more detailed execution phases, which probably will generate more energy savings but with more introduced overhead.

In conclusion, most existing approaches do not expose sufficient information. As a result, formation scarcity of dynamic power dissipation impedes the progress of power-efficient soft-

ware design. The challenge is that there is a gap between the power dissipation of hardware and the applications running on it. In order to design more power/energy efficient systems, both software and hardware need to work in a close loop, which adaptively alter itself according to demand of the other part.

BIBLIOGRAPHY

[1] Pin-A Binary Instrumentation Tool. `http://www.pintool.org/`, Aug 2010.

[2] ACPI. Advanced Configuration Power Interface. `http://www.acpi.info/`.

[3] I. Ahmad, S. Ranka, and S.U. Khan. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –6, apr. 2008.

[4] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[5] Omid Assare and Maziar Goudarzi. Opportunities for embedded software power reductions. In *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on*, pages 000763 –000766, may 2011.

[6] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pages 95 – 105, dec. 2001.

[7] Raid Zuhair Ayoub, Umit Ogras, Eugene Gorbatov, Yanqin Jin, Timothy Kam, Paul Diefenbaugh, and Tajana Rosing. Os-level power minimization under tight performance constraints in general purpose systems. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, ISLPED '11, pages 321–326, Piscataway, NJ, USA, 2011. IEEE Press.

[8] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, December 2007.

[9] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato. A simple power-aware scheduling for multicore systems when running real-time applications. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –7, april 2008.

[10] Frank Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, EW 9, pages 37–42, New York, NY, USA, 2000. ACM.

[11] Luca Benini and Giovanni de Micheli. System-level power optimization: techniques and tools. *ACM Trans. Des. Autom. Electron. Syst.*, 5:115–192, April 2000.

[12] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, pages 147–158, New York, NY, USA, 2010. ACM.

[13] Suparna Bhattacharya, Karthick Rajamani, K. Gopinath, and Manish Gupta. The interplay of software bloat, hardware energy proportionality and system bottlenecks. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 1:1–1:5, New York, NY, USA, 2011. ACM.

[14] Christian Bienia and Kai Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[15] W.L. Bircher, M. Valluri, J. Law, and L.K. John. Runtime identification of microprocessor energy saving opportunities. In *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, pages 275 – 280, aug. 2005.

[16] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[17] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94, New York, NY, USA, 2000. ACM.

[18] David J. Brown and Charles Reams. Toward energy-efficient computing. *Commun. ACM*, 53:50–58, March 2010.

[19] Randal E. Bryant and David R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. Addison-Wesley Publishing Company, USA, 2nd edition, 2010.

[20] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, pages 294 –295, 466, 2000.

[21] Ting Cao, Stephen M Blackburn, Tiejun Gao, and Kathryn S McKinley. The yin and yang of power and performance for asymmetric hardware and managed software. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 225–236, Washington, DC, USA, 2012. IEEE Computer Society.

[22] C.D. Carothers, K.S. Perumalla, and R.M. Fujimoto. The effect of state-saving in optimistic simulation on a cache-coherent non-uniform memory access architecture. In *Simulation Conference Proceedings, 1999 Winter*, volume 2, pages 1624 –1633 vol.2, 1999.

[23] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[24] M. Castro, L.F.W. Goes, C.P. Ribeiro, M. Cole, M. Cintra, and J.-F. Mehaut. A machine learning-based approach for thread mapping on transactional memory applications. In *High Performance Computing (HiPC), 2011 18th International Conference on*, pages 1 –10, dec. 2011.

[25] Fay Chang, Keith Farkas, and Parthasarathy Ranganathan. Energy-driven statistical sampling: Detecting software hotspots. In Babak Falsafi and T. Vijaykumar, editors, *Power-Aware Computer Systems*, volume 2325 of *Lecture Notes in Computer Science*, pages 105–108. Springer Berlin/Heidelberg, 2003.

[26] Jianwei Chen, Michel Dubois, and Per Stenström. Simwattch: Integrating complete-system and user-level performance and power simulators. *IEEE Micro*, 27(4):34–48, 2007.

[27] Ming Chen, Xiaorui Wang, and Xue Li. Coordinating processor and main memory for efficientserver power control. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 130–140, New York, NY, USA, 2011. ACM.

[28] G. Contreras and M. Martonosi. Power prediction for intel xscale reg; processors using performance monitoring unit events. In *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, pages 221 – 226, aug. 2005.

[29] International Business Machines Corporation. IBM PowerExecutive. `http://www-03.ibm.com/systems/management/director/about/director52/extensions/powerexec.html`.

[30] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international con-*

ference on Parallel architectures and compilation techniques, PACT '08, pages 250–259, New York, NY, USA, 2008. ACM.

[31] David Levinthal. *Performance Analysis Guide for Intel® Core™ i7 Processor and Intel® Xeon™ 5500 processors*. 2010.

[32] Willem de Bruijn, Dan Truong, Jesse Brandeburg, Frederic Weisbecker, Carlos R. Mafra, and Pekka Enberg. Perf Wiki. `https://perf.wiki.kernel.org/`, Sep 2009.

[33] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. Memscale: active low-power modes for main memory. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS XVI, pages 225–238, New York, NY, USA, 2011. ACM.

[34] Gaurav Dhiman, Kresimir Mihic, and Tajana Rosing. A system for online power prediction in virtualized environments using gaussian mixture models. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 807–812, New York, NY, USA, 2010. ACM.

[35] Thanh Do, Suhib Rawshdeh, and Weisong Shi. pTop: A Process-level Power Profiling Tool. In *HotPower '09: Proceedings of the Workshop on Power Aware Computing and Systems*, New York, NY, USA, October 2009. ACM.

[36] S.J. Eggers and T.E. Jeremiassen. *Eliminating false sharing*. University of Washington, Department of Computer Science, 1990.

[37] Noel Eisley, Vassos Soteriou, and Li-Shiuan Peh. High-level power analysis for multicore chips. In *CASES '06: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 389–400, New York, NY, USA, 2006. ACM.

[38] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Power challenges may end the multicore era. *Commun. ACM*, 56(2):93–102, February 2013.

[39] Xiaobo Fan, Carla S. Ellis, and Alvin R. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. In *In Workshop on Power-Aware Computing Systems*, pages 164–179, 2002.

[40] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[41] L.M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1548 –1557 vol.3, 2001.

[42] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[43] Ashok Gautham, Kunal Korgaonkar, Patanjali Slpsk, Shankar Balachandran, and Kamakoti Veezhinathan. The implications of shared data synchronization techniques on multi-core energy efficiency. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, HotPower'12, pages 6–6, Berkeley, CA, USA, 2012. USENIX Association.

[44] R. Ge and K.W. Cameron. Power-aware speedup. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1 –10, march 2007.

[45] Rong Ge, Xizhou Feng, and K.W. Cameron. Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1 –8, may 2009.

[46] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):658–671, 2010.

[47] Andy Georges, Dries Buytaert, Lieven Eeckhout, and Koen De Bosschere. Method-level phase behavior in java workloads. In *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '04, pages 270–287, New York, NY, USA, 2004. ACM.

[48] Ravi A. Giri and Anand Vanchi. Increasing data center efficiency with server power measurements. Technical report, Intel Information Technology, 2010.

[49] B. Goel, S.A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. In *Green Computing Conference, 2010 International*, pages 135 –146, aug. 2010.

[50] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithm for dynamic speed-setting of a low-power cpu. In *Proceedings of the 1st annual international conference on Mobile computing and networking*, MobiCom '95, pages 13–25, New York, NY, USA, 1995. ACM.

[51] The Green Grid. The Green Grid Data Center Power Efficiency Metrics: Power Usage Effectiveness and DCiE. `http://www.thegreengrid.org`, 2007.

[52] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, N. Vijaykrishnan, Mahmut Kandemir, Tao Li, and Lizy Kurian John. Using complete machine simulation

for software power estimation: The softwatt approach. In *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, page 141, Washington, DC, USA, 2002. IEEE Computer Society.

[53] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Drpm: dynamic speed control for power management in server class disks. *SIGARCH Comput. Archit. News*, 31(2):169–181, 2003.

[54] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 37–47, New York, NY, USA, 2010. ACM.

[55] Inki Hong, D. Kirovski, Gang Qu, M. Potkonjak, and M.B. Srivastava. Power optimization of variable voltage core-based systems. In *Design Automation Conference, 1998. Proceedings*, pages 176 – 181, jun 1998.

[56] Inki Hong, Miodrag Potkonjak, and Mani B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, ICCAD '98, pages 653–656, New York, NY, USA, 1998. ACM.

[57] S. Hong, S.H.K. Narayanan, M. Kandemir, and O. Ozturk. Process variation aware thread mapping for chip multiprocessors. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 821 –826, april 2009.

[58] Timo Hönig, Christopher Eibel, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Seep: exploiting symbolic execution for energy-aware programming. In *Proceedings*

of the 4th Workshop on Power-Aware Computing and Systems, HotPower '11, pages 4:1–4:5, New York, NY, USA, 2011. ACM.

[59] Chung-Hsing Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. In *Low Power Electronics and Design, International Symposium on, 2001.*, pages 275 –278, 2001.

[60] Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, PLDI '03, pages 38–48, New York, NY, USA, 2003. ACM.

[61] Chunling Hu, Daniel A. Jimenez, and Ulrich Kremer. Toward an evaluation infrastructure for power and energy optimizations. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11*, page 230.2, Washington, DC, USA, 2005. IEEE Computer Society.

[62] Clay Hughes and Tao Li. Optimizing throughput/power trade-offs in hardware transactional memory using dvfs and intelligent scheduling. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 141–150, New York, NY, USA, 2011. ACM.

[63] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Number 253669-033US. December 2009.

[64] Intelligent Platform Management Interface. `http://www.intel.com/design/servers/ipmi/index.htm`.

[65] C. Isci and M. Martonosi. Phase characterization for power: evaluating control-flow-based and event-counter-based techniques. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 121 – 132, feb. 2006.

[66] Canturk Isci, Gilberto Contreras, and Margaret Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 359 –370, dec. 2006.

[67] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, Washington, DC, USA, 2003. IEEE Computer Society.

[68] Bruce Jacob, Spencer W. Ng, and David T. Wang. *Memory Systems : Cache, DRAM, Disk*, pages 43–44. Denise E.M. Penrose, 2007.

[69] José A. Joao, M. Aater Suleman, Onur Mutlu, and Yale N. Patt. Bottleneck identification and scheduling in multithreaded applications. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 223–234, New York, NY, USA, 2012. ACM.

[70] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A survey of energy efficient network protocols for wireless networks. *Wirel. Netw.*, 7:343–358, September 2001.

[71] Russ Joseph, David Brooks, and Margaret Martonosi. Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs. In *In Workshop on Complexity Effectice Design WCED, held in conjunction with ISCA-28. Jun 2001*, June 2001.

[72] Russ Joseph and Margaret Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, pages 135–140, New York, NY, USA, 2001. ACM.

[73] Da-Cheng Juan and Diana Marculescu. Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, ISLPED '12, pages 97–102, New York, NY, USA, 2012. ACM.

[74] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE '01, pages 28–31, New York, NY, USA, 2001. ACM.

[75] S. Kamil, J. Shalf, and E. Strohmaier. Power efficiency in high performance computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –8, Apr. 2008.

[76] Aman Kansal and Feng Zhao. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.*, 36:26–31, August 2008.

[77] S.U. Khan and I. Ahmad. A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. *Parallel and Distributed Systems, IEEE Transactions on*, 20(3):346 –360, march 2009.

[78] Minjang Kim, P. Kumar, Hyesoon Kim, and B. Brett. Predicting potential speedup of serial code via lightweight profiling and emulations with memory performance model. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1318 –1329, may 2012.

[79] T. Kitahara, F. Minami, T. Ueda, K. Usami, S. Nishio, M. Murakata, and T. Mitsuhashi. A clock-gating method for low-power lsi design. In *Design Automation Conference 1998. Proceedings of the ASP-DAC '98. Asia and South Pacific*, pages 307 –312, feb 1998.

[80] M. Lammie, P. Brenner, and D. Thain. Scheduling grid workloads on multicore clusters to minimize energy and maximize performance. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 145 –152, oct. 2009.

[81] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XII, pages 185–194, New York, NY, USA, 2006. ACM.

[82] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 2–13, New York, NY, USA, 2009. ACM.

[83] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2):183–195, 2008.

[84] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, 2003.

[85] Dake Liu and C. Svensson. Power consumption estimation in cmos vlsi chips. *Solid-State Circuits, IEEE Journal of*, 29(6):663 –670, jun. 1994.

[86] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flikker: saving dram refresh-power through critical data partitioning. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS XVI, pages 213–224, New York, NY, USA, 2011. ACM.

[87] Jacob R. Lorch and Alan J Smith. Energy consumption of apple macintosh computers. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1997.

[88] J.R. Lorch and A.J. Smith. Software strategies for portable computer energy management. *Personal Communications, IEEE*, 5(3):60 –73, jun 1998.

[89] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Power-aware operating systems for interactive systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 10(2):119–134, 2002.

[90] Aqeel Mahesri and Vibhore Vardhan. Power consumption breakdown on a modern laptop. In Babak Falsafi and T. VijayKumar, editors, *Power-Aware Computer Systems*, volume 3471 of *Lecture Notes in Computer Science*, pages 165–180. Springer Berlin / Heidelberg, 2005.

[91] Diana Marculescu, Radu Marculescu, and Massoud Pedram. Information theoretic measures of energy consumption at register transfer level. In *ISLPED '95: Proceedings of the 1995 international symposium on Low power design*, pages 81–86, New York, NY, USA, 1995. ACM.

[92] Thomas L. Martin and Daniel P. Siewiorek. Nonideal battery and main memory effects on cpu speed-setting for low power. *IEEE Trans. Very Large Scale Integr. Syst.*, 9:29–34, February 2001.

[93] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: eliminating server idle power. *SIGPLAN Not.*, 44:205–216, March 2009.

[94] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *Proceeding of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 319–330, New York, NY, USA, 2011. ACM.

[95] Wattsup Meters. Watts up? .net. `https://www.wattsupmeters.com`.

[96] Dennis Mocigemba. Sustainable computing. *Poiesis & Praxis: International Journal of Technology Assessment and Ethics of Science*, 4:163–184, 2006. 10.1007/s10202-005-0018-8.

[97] D. Molaro, H. Payer, and D. Le Moal. Tempo: Disk drive power consumption characterization and modeling. In *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on*, pages 246 –250, may 2009.

[98] Luca Niccolini, Gianluca Iannaccone, Sylvia Ratnasamy, Jaideep Chandrashekar, and Luigi Rizzo. Building a power-proportional software router. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX ATC'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.

[99] Jaewon Oh and M. Pedram. Gated clock routing for low-power microprocessor design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(6):715 –722, jun 2001.

[100] C.S. Patel, S.M. Chai, S. Yalamanchili, and D.E. Schimmel. Power constrained design of multiprocessor interconnection networks. In *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, pages 408 –416, oct 1997.

[101] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets '11, pages 5:1–5:6, New York, NY, USA, 2011. ACM.

[102] Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of*

the sixth conference on Computer systems, EuroSys '11, pages 153–168, New York, NY, USA, 2011. ACM.

[103] A. Peymandoust, T. Simunic, and G. de Micheli. Low power embedded software optimization using symbolic algebra. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '02, pages 1052–, Washington, DC, USA, 2002. IEEE Computer Society.

[104] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 89–102, New York, NY, USA, 2001. ACM.

[105] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 251–259, New York, NY, USA, 2001. ACM.

[106] M.D. Powell, A. Biswas, J.S. Emer, S.S. Mukherjee, B.R. Sheikh, and S. Yardi. Camp: A technique to estimate per-structure power at run-time using a few simple parameters. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 289 –300, feb. 2009.

[107] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson. Application-aware power management. In *Workload Characterization, 2006 IEEE International Symposium on*, pages 39 –48, oct. 2006.

[108] Rambus Incorporated. Rambus. `http://www.rambus.com/us/`.

[109] S. Ramprasad, N.R. Shanbhag, and I.N. Hajj. Signal coding for low power: fundamental limits and practical realizations. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 46(7):923 –929, jul 1999.

[110] Parthasarathy Ranganathan. Recipe for efficiency: principles of power-aware computing. *Commun. ACM*, 53:60–67, April 2010.

[111] Andy Rawson, John Pflueger, and Tahir Cader. The green grid data center power efficiency metrics: Pue and dcie. Technical report, The Green Grid, October 2007.

[112] S. Reda. Thermal and power characterization of real computing devices. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 1(2):76 –87, june 2011.

[113] Arjun Roy, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 139–152, New York, NY, USA, 2011. ACM.

[114] Jeffry T. Russell and Margarida F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *ICCD '98: Proceedings of the International Conference on Computer Design*, page 328, Washington, DC, USA, 1998. IEEE Computer Society.

[115] Vijayalakshmi Saravanan, Senthil Kumar Chandran, Sasikumar Punnekkat, and D. P. Kothari. A study on factors influencing power consumption in multithreaded and multicore cpus. *W. Trans. on Comp.*, 10(3):93–103, March 2011.

[116] Eric Saxe. Power-efficient software. *Commun. ACM*, 53(2):44–48, 2010.

[117] Robert Schöne, Daniel Hackenberg, and Daniel Molka. Simultaneous multithreading on x86_64 systems: an energy efficiency evaluation. In *Proceedings of the 4th Workshop*

on Power-Aware Computing and Systems, HotPower '11, pages 10:1–10:5, New York, NY, USA, 2011. ACM.

[118] Robert Schöne, Daniel Hackenberg, and Daniel Molka. Memory performance at reduced cpu clock speeds: an analysis of current x86_64 processors. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, HotPower'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[119] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 29 – 40, feb. 2002.

[120] Navin Sharma, Sean Barker, David Irwin, and Prashant Shenoy. Blink: managing server clusters on intermittent power. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS XVI, pages 185–198, New York, NY, USA, 2011. ACM.

[121] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. *SIGOPS Oper. Syst. Rev.*, 36:45–57, October 2002.

[122] Youngsoo Shin and Kiyoung Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 134 –139, 1999.

[123] Karan Singh, Major Bhadauria, and Sally A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37:46–55, July 2009.

[124] A. Sinha, N. Ickes, and A.P. Chandrakasan. Instruction level and operating system profiling for energy exposed software. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(6):1044 –1057, dec. 2003.

[125] Ecos Plug Load Solutions. 80plus.org. `http://www.80plus.org`, 2008.

[126] CA) Song, Seungyoon Peter (Palo Alto. Method and system for selective DRAM refresh to reduce power consumption. `http://www.freepatentsonline.com/6094705.html`, July 2000.

[127] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1 –8, july 2011.

[128] Bob Steigerwald, Christopher D. Lucero, Chakravarthy Akella, and Abhishek Agrawal. *Energy Aware Computing Powerful Approaches for Green System Design*. Intel Press, March 2012.

[129] T.K. Tan, A. Raghunathan, G. Lakshminarayana, and N.K. Jha. High-level software energy macro-modeling. In *Design Automation Conference, 2001. Proceedings*, pages 605 – 610, 2001.

[130] The Standard Performance Evaluation Corporation (SPEC). SPECjvm2008 V1.01. `http://www.spec.org/download.html`, Jun 2009.

[131] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. Very Large Scale Integr. Syst.*, 2(4):437–445, 1994.

[132] J. Treibig, G. Hager, and G. Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 207 –216, sept. 2010.

[133] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48 – 56, may 2005.

[134] Amin Vahdat, Alvin Lebeck, and Carla Schlatter Ellis. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 31–36, New York, NY, USA, 2000. ACM.

[135] Marc A. Viredaz, Marc A. Viredaz, Deborah A. Wallach, and Deborah A. Wallach. Power evaluation of itsy version 2.3. Technical report, Compaq, Western Research Laboratory, 2000.

[136] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, pages 294 – 305, 2002.

[137] Shinan Wang, Hui Chen, and Weisong Shi. SPAN: A software power analyzer for multicore computer systems. *Sustainable Computing: Informatics and Systems*, 1(1):23 – 34, 2011.

[138] Shinan Wang and Weisong Shi. Cpt: An energy efficiency model for multi-core computer systems. Technical Report MIST-TR-2012-008, Wayne State University, October 2012.

[139] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[140] Wikipedia. App Store (iOS). {http://en.wikipedia.org/wiki/App_Store_(iOS)}.

[141] Donghong Wu, Bingsheng He, Xueyan Tang, Jianliang Xu, and Minyi Guo. Ramzzz: rank-aware dram power management with dynamic migrations and demotions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 32:1–32:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[142] Q. Wu, M. Pedram, and X. Wu. Clock-gating and its application to low power design of sequential circuits. In *Custom Integrated Circuits Conference, 1997., Proceedings of the IEEE 1997*, pages 479 –482, may 1997.

[143] Wei Wu, Lingling Jin, Jun Yang, Pu Liu, and Sheldon X.-D. Tan. A systematic method for functional unit power estimation in microprocessors. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 554–557, New York, NY, USA, 2006. ACM.

[144] Erlin Yao, Yungang Bao, Guangming Tan, and Mingyu Chen. Extending amdahl's law in the multicore era. *SIGMETRICS Perform. Eval. Rev.*, 37(2):24–26, October 2009.

[145] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 374–, Washington, DC, USA, 1995. IEEE Computer Society.

[146] Terry Tao Ye, Giovanni De Micheli, and Luca Benini. Analysis of power consumption on switch fabrics in network routers. In *DAC '02: Proceedings of the 39th annual Design Automation Conference*, pages 524–529, New York, NY, USA, 2002. ACM.

[147] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *DAC '00: Proceedings of the 37th*

Annual Design Automation Conference, pages 340–345, New York, NY, USA, 2000. ACM.

[148] M.T. Yourst. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 23–34, 2007.

[149] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[150] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: managing energy as a first class operating system resource. *SIGPLAN Not.*, 37(10):123–132, 2002.

[151] Dimitrios Ziakas, Allen Baum, Robert A. Maddox, and Robert J. Safranek. Intel®; quickpath interconnect architectural features supporting scalable system architectures. In *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, HOTI '10, pages 1–6, Washington, DC, USA, 2010. IEEE Computer Society.

ABSTRACT

SOFTWARE POWER ANALYSIS AND OPTIMIZATION FOR POWER-AWARE MULTICORE SYSTEMS

by

SHINAN WANG

May 2014

Advisor: Dr. Weisong Shi

Major: Computer Science

Degree: Doctor of Philosophy

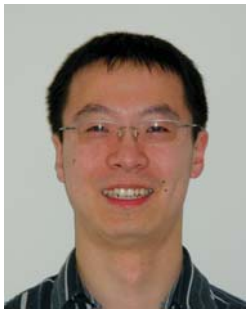
Among all the factors in sustainable computing, power dissipation and energy consumption, arguably speaking, are fundamental aspects of modern computer systems. Different from performance metric, power dissipation is not easy to measure because hardware instrumentation is usually required. Yet as an indispensable component of a computer system, software becomes a major factor affecting power dissipation besides hardware energy-efficiency and power states. With detailed information on resource usage and power dissipation of an application/software, software developers will be able to leverage algorithms and implementations in order to produce power-efficient solutions. Hardware instrumentation, despite its accuracy, is costly and complicated to set up. A general solution to connect software with hardware along with detailed power and system information will improve the system overall efficiency.

In this work, we design and implement a general solution to analyze and model software power dissipation. Based on the analysis, we propose a combined solution to optimize the energy efficiency of parallel workload. Starting from the hands-on power measurement method in detail, we provide a fine-grain power profile of two computer systems using hardware instrumentation. Being focusing on dynamic power dissipation analysis, we propose a two-level power model for power-aware multicore computer systems. Based on the model, we design and implement SPAN to relate power dissipation to the different portions of an application

using the proposed power model. By using SPAN, developers can easily identify the sections of code consuming the most power in the program. Alternatively, to enable automatic source code instrumentation, we utilize compiler techniques to insert profiling code before and after each function in source code. The expected outcome includes an open source function level power profiling tool, Safari. Using the profiling tools, we propose a model to capture the relationship between concurrency (C), power (P) and execution time (T). By changing the system configuration for different parallel workload, we are able to achieve optimal/near optimal energy-efficient execution of a given workload on a specific platform.

AUTOBIOGRAPHICAL STATEMENT

SHINAN WANG



Shinan Wang joined Ph.D program at Wayne State University in 2008. He received his Bachelor degrees in Electronic Engineering from Beijing University of Posts and Telecommunications, China, 2008. His research interests include Green Computing, High Performance Computing, Computer Architecture, and . He has published several papers in workshops, conferences and journal, such as Suscom, ICEAC, WEED. He has also served as a peer reviewer for many conferences, such as CollaborateCom etc.